UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA


# OTIMIZAÇÃO DA DESCENTRALIZAÇÃO DE PROCESSOS DE NEGÓCIO DEPENDENTES DE IOT

**Miguel Filipe Saragoça Espírito Santo**

**Mestrado em Engenharia Informática**
Especialização em Sistemas de Informação

Dissertação orientada por:
Profª. Doutora Ana Luísa do Carmo Correia Respício
Profª. Doutora Maria Dulce Pedroso Domingos

2020

# Agradecimentos

Agradeço o apoio prestado pela FCT – Fundação para a Ciência e a Tecnologias, através do projeto DOIT, com a referência PTDC/EEI-ESS/5863/2014 e à Unidade de Investigação LASIGE, com a referência UIDB/00408/2020.

Obrigado à professora Ana e à professora Dulce por toda a ajuda e paciência que tiveram comigo ao longo de todo este tempo.

Um especial agradecimento à minha família por todo o apoio que me têm dado. Aos meus avós, que nunca escondiam a felicidade por me verem voltar a Évora aos fins de semana, e pela saudade que ficava quando regressava a Lisboa para mais uma semana de aulas. Aos meus pais, um obrigado não chega. Por toda a confiança que depositaram em mim, e o apoio incondicional que me deram desde o primeiro dia. Tudo aquilo que sou hoje devo a eles.

Obrigado aos meus amigos de Évora, que tive a sorte de me acompanharem na vinda para Lisboa, desde 2013. Sem eles, a experiência nunca teria sido a mesma. A todas as pessoas incríveis que tive o privilégio de conhecer desde o primeiro dia de FCUL. Fossem eles de Lisboa, Porto, Santarém ou Machico. Marcaram-me.

Obrigado ao Daniel Santos e ao Marcus Dias por toda a ajuda que me deram ao longo de todo o curso. Pela confiança que depositaram em mim ao escolher-me para tantos dos projetos em grupo, ou pelo tempo que investiram em ajudar-me a estudar para os exames. Este título também é vosso.

Por fim, André Carocha e Micael Franco: vocês já sabem. Não dá para contabilizar todos os momentos que passámos desde as primeiras idas à praxe até aos dias de hoje: os fins de tarde ao sol, jantares, noitadas, sessões de estudo, treinos, férias, festas… que assim continue por muito mais tempo.

*Aos meus pais*

# Resumo

As redes *IoT* (*Internet of Things*) assumem cada vez mais preponderância na automação de diferentes processos do dia a dia, e o seu impacto a vários níveis na nossa vida é uma realidade. Estas caracterizam-se principalmente pelo uso de dispositivos de *hardware*, nomeadamente sensores, que podem recolher e transmitir vários tipos de informação, conforme a finalidade desejada pelas organizações. A informação é depois utilizada na automação de processos em diversos setores da nossa sociedade, como transportes, cidades inteligentes, e saúde, por exemplo. Em todos estes exemplos, fatores como a fiabilidade são aspetos fulcrais a garantir nestas redes. De forma a que estas cumpram com a finalidade pretendida, o seu funcionamento recai em conjuntos estruturados de atividades com uma ordem específica de execução, os quais se designam de processos de negócio.

Grande parte das redes *IoT* atuais apresentam ainda hoje uma arquitetura centralizada, onde os sensores enviam a informação recolhida de um dado ambiente para um sistema central. O sistema central realiza, posteriormente, tomadas de decisão com base na informação recolhida, tais como enviar uma ordem de realização de uma tarefa a um atuador. No entanto, vários custos resultam deste tipo de arquitetura. Um dos mais dispendiosos deve-se ao consumo de energia por parte dos sensores, resultante da comunicação a que são sujeitos com o sistema central.

A descentralização dos processos toma partido das capacidades de processamento dos dispositivos *IoT*, com o intuito de se reduzir a comunicação efetuada por estes. Essa medida resulta em alternativas com um menor custo derivado da comunicação, pois haverá um menor número de mensagens transmitidas em todo o sistema. Em contrapartida, a fiabilidade do processo poderá ser afetada, para melhor ou para pior. É, por isso, necessário estabelecer um equilíbrio entre uma fiabilidade elevada e um custo reduzido da comunicação do processo. Neste trabalho é definido um problema de otimização que pretende encontrar as alternativas que melhor cumprem esse equilíbrio. Para tal são implementados dois tipos de métodos para resolver o problema em questão: um de busca exaustiva e dois métodos da classe de algoritmo meta-heurísticos.

**Palavras-chave:** Internet das Coisas, Processos de negócio, BPMN, Otimização multi-objetivo, Algoritmos meta-heurísticos

# Abstract

IoT (Internet of Things) networks are becoming increasingly prevalent in the automation of different day-to-day processes, and their impact at various levels in our lives is a reality. These are mainly characterized by using hardware devices, namely sensors, which can collect and transmit various types of information, according to the purpose desired by organizations. The information is then used to automate processes in various sectors of our society, such as transports, smart cities, and health, for instance. In all these examples, factors such as reliability are key aspects to be guaranteed in these networks. For them to fulfil the intended purpose, their operation falls into structured sets of activities with a specific order of execution, which are called business processes.

Most of today's IoT networks still have a centralized architecture, where sensors send information collected from a given environment to a central system. The central system subsequently makes decisions based on the information collected, such as sending an order to perform a task to an actuator. However, several costs result from this type of architecture. One of the most expensive is due to the energy consumption by the sensors, resulting from the communication to which they are subject with the central system.

The decentralization of processes takes advantage of the processing capabilities of IoT devices, to reduce the communication made by them. This measure results in alternatives with a lower cost derived from communication, as there will be a smaller number of messages transmitted throughout the system. On the other hand, the reliability of the process may be affected, for better or for worse. It is therefore necessary to strike a balance between high reliability and a low cost of communications in the process. In this work, an optimization problem is defined that seeks to find the alternatives that best achieve this balance. For this, two types of methods are implemented to solve the problem in question: one of exhaustive search and two methods of the meta-heuristic algorithm class.

**Keywords:** *Internet of Things*, Business Processes, BPMN, Multi-objective optimization, Meta-heuristic algorithms

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1 Introdução

Este capítulo descreve a motivação deste trabalho, os objetivos com ele pretendidos, as contribuições prestadas e a organização do documento.

## 1.1 Motivação

A importância da Internet das Coisas, ou IoT (*Internet of Things*), tem vindo a aumentar cada vez mais no mundo da informação e comunicação. Os setores da indústria passaram a adotar a realização de atividades automáticas por meio de aparelhos eletrónicos que podem ter diversas finalidades. Esses aparelhos encontram-se aptos a receber, enviar ou processar informação de forma a automatizar processos de negócio, e de certa forma contribuir para o avanço da sociedade em várias áreas. Sensores e atuadores são os exemplos mais conhecidos de dispositivos utilizados nas redes IoT. Uma rede composta por tais aparelhos pode ser útil, por exemplo, no desenvolvimento de cidades inteligentes, contribuindo para a melhoria do tráfego, abastecimento da população, sistemas de saúde, segurança, etc – todo um vasto conjunto de meios dispostos a melhorar a qualidade de vida das populações [20].

As redes IoT apresentam diversos desafios inerentes ao seu design e desenvolvimento. Este tipo de sistemas pode apresentar uma vasta heterogeneidade de elementos, como sensores e atuadores, e garantir que cada um cumpre a sua função, bem como assegurar a correta comunicação entre esses, são ações fundamentais. Fatores como o consumo de energia dos dispositivos, escalabilidade e fiabilidade dos sistemas são dos principais aspetos a ter em conta na projeção deste tipo de redes [1]. Em sistemas de saúde, por exemplo, a fiabilidade [25] e segurança são aspetos considerados imperativos a todo o momento, pois uma única falha pode apresentar riscos para vidas humanas que se encontrem em constante monitorização.

Por norma, os sensores, ao recolherem informação do ambiente onde estão inseridos, enviam-na para um sistema central responsabilizado pelo seu processamento, bem como pela tomada de decisões lógicas inseridas nos processos de negócio. A constante comunicação entre um sensor e um sistema central apresenta, contudo, desvantagens nomeadamente ao nível do consumo de bateria do sensor. Sobretudo em sistemas de média e grande escala, a manutenção dos dispositivos pode tornar-se

impraticável. Pretende-se, por isso, adotar uma alternativa de computação descentralizada [18] que permita:

- Um aumento significativo de processamento a ser realizado por parte dos aparelhos IoT envolvidos;
- Um aumento de vida útil da bateria dos aparelhos; e
- Alívio de carga no sistema central, derivada do envio e resposta a pedidos aos quais este é sujeito.

Dadas as áreas sensíveis em que os sistemas *IoT* são utilizados, tais como sistemas de saúde, é fundamental garantir-se que o número de falhas é tão pequeno quanto possível, idealmente perto de zero. Ou seja, estes sistemas devem apresentar uma fiabilidade muito elevada, independentemente de a computação estar ou não descentralizada.

Os processos executados por sistemas IoT são desenhados *a priori* e para esse desenho recorre-se muitas vezes a uma modelação baseada em processos de negócio. Em particular, esses processos são referidos na literatura como processos de negócio dependentes de *IoT* [7].

Para desenhar estes processos é comum o uso da notação Business Process Model and Notation (BPMN), que é a linguagem padrão utilizada na modelação de processos de negócio [21]. Esta notação permite representar e visualizar os processos graficamente sob forma de *workflows*, identificando de forma clara os elementos que os constituem e as interações entre eles.

Tendo em conta o exposto anteriormente, é na fase de desenho do ciclo de vida BPM que devemos ter a preocupação de descentralizar a computação nos processos de negócio dependentes de *IoT*.

## 1.2 Objetivos

O objetivo deste trabalho é desenvolver métodos que permitam otimizar a descentralização da computação em processos de negócio dependentes de IoT, modelados com notação *Business Process Model and Notation* (BPMN), considerando como objetivos a maximização da fiabilidade do processo e a minimização do custo de

comunicação. Para tal, recorre-se a algoritmos meta-heurísticos para problemas de otimização multi-objetivo.

## 1.3 Contribuições

Este trabalho foi realizado no âmbito do projeto DoIT, com a referência PTDC/EEIESS/5863/2014, na Unidade de Investigação LASIGE, com a referência UIDB/00408/2020. A concretização deste trabalho foi possível com recurso a um *software* previamente desenvolvido que permite gerar estruturas de dados que representam processos de negócio modelados em BPMN. A geração pode ser realizada por um método pseudoaleatório ou por tradução de processos elaborados graficamente. Foram realizados testes ao *software*, bem como efetuadas melhorias. Posteriormente, foram implementados algoritmos multi-objetivo que otimizam a descentralização de processos de negócio dependentes de *IoT*. Um dos algoritmos segue um método de busca exaustiva, e os outros dois são baseados nos métodos meta-heurísticos *Local Search* e *Simulated Annealing*. Os algoritmos foram posteriormente avaliados quanto ao seu desempenho, através de um conjunto de métricas, tendo-se observado uma boa qualidade dos resultados.

## 1.4 Organização do documento

O documento encontra-se organizado pela seguinte ordem:

- Capítulo 2 – Apresenta os conceitos relacionado. Elementos e propriedades das redes IoT. Notação BPMN, integração de aspetos de Qualidade de Serviço (QoS) na notação, e decomposição de processos de negócio dependentes de *IoT*. Otimização Combinatória e Meta-heurísticas.
- Capítulo 3 – Apresenta o trabalho realizado. Definição do problema de otimização, considerando os objetivos de fiabilidade e custo da comunicação. Elaboração de algoritmos de otimização.
- Capítulo 4 – Geração de processos de exemplo, e execução de testes sob esses processos com recurso aos algoritmos meta-heurísticos e de busca exaustiva elaborados.
- Capítulo 5 – Conclusões e possíveis melhorias.

# Capítulo 2 Conceitos relacionados

Neste capítulo é realizada uma breve introdução dos conceitos usados no trabalho. É descrito o paradigma da Internet das Coisas, nomeadamente as camadas da arquitetura e as propriedades intrínsecas do sistema. É introduzida a linguagem BPMN e principais elementos que a constituem. São referidos dois exemplos de processos de negócio dependentes de *IoT*, modelados em BPMN. A área da otimização combinatória é introduzida, bem como o uso de algoritmos meta-heurísticos.

## 2.1 Internet das coisas (*IoT*)

Embora não exista uma definição formal de *Internet of Things*, são várias as descrições presentes na literatura. Uma delas encontra-se em [13] e classifica *IoT* como "uma infraestrutura de rede global dinâmica com recursos de autoconfiguração baseados em protocolos de comunicação padrão e interoperáveis onde "coisas" físicas e virtuais têm identidades, atributos físicos e personalidades virtuais e usam interfaces inteligentes e estão perfeitamente integradas na rede de informações".

*IoT* é também definido em [30] como "uma infraestrutura global para a sociedade da informação, que permite serviços avançados ao interligar coisas (físicas e virtuais) com base em tecnologias de informação e comunicação interoperáveis, existentes e em evolução."

### 2.1.1 Arquitetura *IoT*

A arquitetura das redes *IoT* é tipicamente caracterizada por quatro camadas principais. A camada física (ou de perceção) é o nível mais simples da arquitetura, e é representada pelos vários aparelhos que comunicam através da camada de rede. O *middleware* realiza o processamento da informação recolhida e mantém-na disponível para as aplicações. Como em todas camadas existem riscos de segurança devem-se aplicar controlos de prevenção de ataques em cada uma delas. De seguida, é apresentado cada nível da arquitetura [16], bem como a sua representação na Figura 2.1.

- **Camada de perceção**: é constituída por todos os aparelhos físicos da rede responsáveis por detetar informação ou executar tarefas no ambiente onde estão inseridos. Valores de temperatura e humidade de um local, e rastreio da

localização e movimentos de pessoas e objetos são exemplos de informações que podem ser recolhidas. Entre as principais tecnologias encontram-se as *Radio Frequency Identification* (*RFID*), *Wireless Sensor Networks* (*WSNs*) e *Global Positioning System* (*GPS*). Aparelhos como sensores, *tags* RFID, atuadores ou *smartphones* encontram-se geralmente entre os mais utilizados.

- **Camada de rede**: é responsável por transmitir de forma fiável a informação da camada de perceção para um sistema onde esta possa ser processada. A transmissão dos dados é realizada através de redes de comunicação, tais como a internet, redes móveis, etc.
- **Camada de suporte (*middleware*)**: a este nível, a informação enviada pela camada de rede é transformada, podendo recorrer a tecnologias no âmbito do processamento analítico, bem como de computação em nuvem. Os dados são armazenados de forma permanente para serem usados quando necessário.
- **Camada de aplicação**: fornece um serviço que vai de acordo com os requisitos dos utilizadores. Este serviço pode ser apresentado sob forma de interface aos utilizadores, nomeadamente através do uso de dispositivos pessoais como computador ou equipamentos móveis.

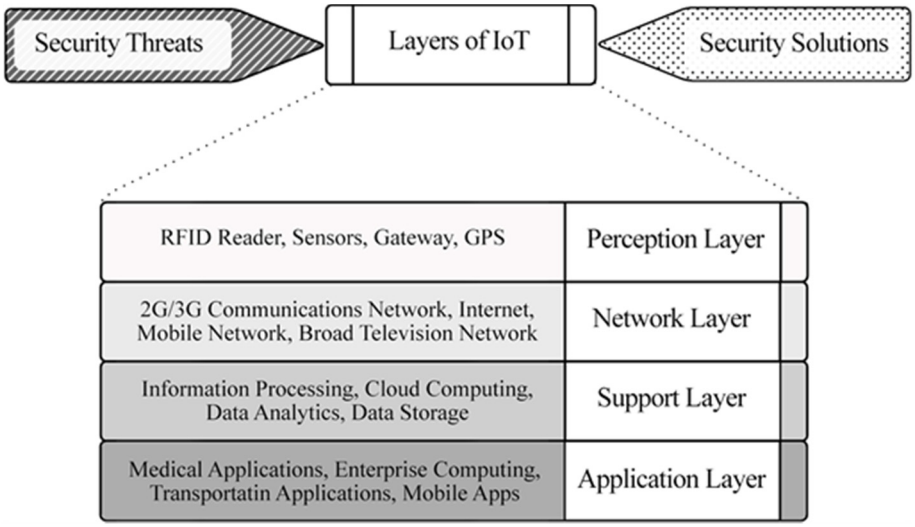


Figura 2.1: Arquitetura IoT, extraída de [16]

### 2.1.2 *IoT* e suas propriedades

O trabalho [20] enumera um conjunto de características inerentes aos sistemas *IoT*, bem como desafios daí resultantes:

- **Heterogeneidade dos aparelhos**: o sistema é constituído por uma grande variedade de aparelhos com diferentes características, tanto em termos de computação como de comunicação.
- **Escalabilidade**: existem vários desafios a ultrapassar à medida que os elementos da rede aumentam. O foco principal deverá ser ao nível do endereçamento, da comunicação (devido à forte correlação entre várias entidades), da gestão da informação, e do fornecimento e gestão da grande variedade de serviços que serão usados.
- **Troca ubíqua de dados em redes sem fios**: as tecnologias de redes sem fios são usadas essencialmente em ambientes pervasivos, onde a troca de dados entre objetos é constante. Deve haver, por isso, um esforço em se garantir a disponibilidade do sistema.
- **Otimização de energia**: o uso de energia por parte dos aparelhos IoT é um recurso dispendioso, principalmente quando usada ao nível a comunicação. Devem ser encontradas soluções que minimizem a quantidade de energia usada, mantendo-se a eficácia do sistema.
- **Capacidades de localização e rastreamento**: várias aplicações fazem uso de tecnologias RFID para rastrear a localização e movimentos dos objetos da rede, como por exemplo ao nível da saúde, logística ou gestão do ciclo de vida do produto.
- **Capacidades de auto-organização**: muitos dos cenários IoT dependem da autonomia de objetos em redes ad hoc, onde não existe uma topologia estabelecida. Os objetos devem ser capazes de se adaptar ao ambiente onde estão inseridos, reorganizando-se entre eles e realizando tarefas coordenadas, evitando tanto quanto possível a necessidade de intervenção humana.
- **Interoperabilidade semântica e gestão de dados**: há uma grande quantidade de dados trocada e analisada por diversos elementos da rede. Como tal, devem ser usados formatos e linguagens bem definidos que permitam aos recetores da comunicação interpretar a informação tal como foi enviada pelos transmissores.

É, por isso, fulcral que as aplicações suportem a inferência correta e autónoma dos dados.

- **Mecanismos embebidos de privacidade e segurança**: devido à estrita ligação que as tecnologias *IoT* têm com o mundo físico, é essencial tomar medidas e definir políticas de privacidade, nomeadamente quando na presença de dados sensíveis. A segurança deve ser assegurada, já que os elementos das redes constituirão uma possível base de ataques maliciosos.

## 2.2 Processos de negócio dependentes de IoT

As redes *IoT* assumem atualmente um papel preponderante na recolha e tratamento de informação. O aumento crescente do uso e complexidade deste tipo de redes nos processos de negócio das empresas é visível, e torna-se necessário o uso de uma ferramenta prática na modelação desses processos. A notação BPMN (*Business Process Model and Notation*) veio ajudar nesse sentido [21]. Esta especificação, que modela e executa processos de negócio, permite identificar e separar claramente os elementos de uma rede quanto à sua função e localização. A notação é composta por diversos elementos, divididos em cinco categorias principais:

- Objetos de fluxo: são os elementos que definem o comportamento de um processo de negócio. Podem incluir eventos (iniciais, intermédios e finais), atividades (tarefas ou subprocessos colapsados) ou *gateways*.
- Dados: fornece informação sobre o que as atividades precisam para serem realizadas, bem como informação produzida por elas.
- Objetos de ligação: definem a ordem de execução dos objetos de fluxo, criando conexões entre eles. Podem ser fluxos de sequência, de mensagem, ou associações, normais ou de dados.
- *Swimlanes*: representam as entidades que executam o processo. Podem ser *pools*, que designam os participantes (por exemplo, uma empresa) e *lanes*, que podem variar no significado (sistema, departamento, etc).
- Artefactos: úteis para fornecer informação adicional.

### 2.2.1 Qualidade de informação e custo dos recursos IoT

Os sensores, bem como outros recursos físicos usados nas redes *IoT*, podem apresentar diferentes características, dependendo do que se procura obter num determinado sistema. Por um lado, um dado sensor pode apresentar uma grande qualidade de informação, mas por outro também é provável que o custo seja mais elevado. Se num determinado sistema for necessário reduzir o custo dos recursos usados, será provável que a qualidade de informação disponível também diminua.

O objetivo passa então por encontrar um equilíbrio entre as duas componentes de forma a se otimizar as funcionalidades do sistema, de acordo com as restrições impostas.

Em [17] é proposta a adição de informação do custo ao nível dos processos de negócio. A informação inclui vários tipos de custo intrínsecos aos sistemas *IoT*, tais como custos de energia, custos de atuação e custos de comunicação. Primeiro, são adicionadas descrições dos recursos ao nível do serviço, e depois ao nível dos processos, ao se estender a linguagem BPMN.

### 2.2.2 Fiabilidade de processos BPMN

Com a modelação de processos de negócio BPMN tornou-se necessário integrar aspetos de Qualidade de Serviço. Estes aspetos pretendem realçar características essenciais ao bom funcionamento de um sistema, tais como o seu custo, fiabilidade, disponibilidade, etc.

A fiabilidade é um dos aspetos mais importantes a ter em conta, e indica-nos o quão provável é o processo ou um elemento do processo realizar sem falhas a atividade que lhe é esperada. Dito isto, se cada elemento único de um dado processo BPMN tiver um valor de fiabilidade associado conhecido à partida, ser-nos-á possível calcular a fiabilidade total desse processo.

Em [24] é proposto um método para calcular a fiabilidade geral de processos BPMN, tendo por base o algoritmo SWR (*Stochastic Workflow Reduction*), destinado a *workflows*. Este algoritmo foi proposto por Cardoso [6] e aplica um conjunto de reduções nos elementos de um *workflow* até restar apenas um elemento (tarefa) que o represente na sua totalidade. As reduções são efetuadas de acordo com seis padrões bem definidos que podem ser encontrados num *workflow*, com vista a converter os elementos de um dado padrão numa só tarefa. Após o conjunto de reduções terminar, deverá permanecer

um único elemento, cuja fiabilidade será equivalente à fiabilidade geral do *workflow* em questão.

É também definido em [6] que o valor de fiabilidade de uma tarefa individual *t* é *R(t)* = 1 - *failureRate(t)*, onde *failureRate(t)* é o rácio *número de execuções perdidas/número de execuções agendadas* da tarefa *t*. Esta abordagem pode então ser aplicada a um processo BPMN, onde cada atividade tem a sua própria fiabilidade já definida, e são identificados os seis padrões possíveis para se reduzir o processo, denominados blocos de processo, cada um com uma fórmula distinta para o cálculo da respetiva fiabilidade. Os blocos em questão podem ser dos seguintes tipos: sequencial, paralelo, condicional, ciclo, tolerante a faltas, e subprocessos.

### 2.2.3 Fiabilidade de recursos em processos BPMN

Até aqui teve-se em conta a fiabilidade de cada atividade individual de um processo, de modo a calcular a fiabilidade total deste. Contudo, não era tida em conta a possível presença de recursos associados à execução das atividades do processo, também esses com um valor de fiabilidade atribuído. Estes recursos podem tanto ser humanos como peças de *hardware*, como por exemplo sensores, ou componentes de *software*.

Em [9] é apresentada uma proposta de integração desses recursos, de forma a que as fiabilidades desses recursos sejam usadas no cálculo da fiabilidade total do processo. A integração dos recursos é realizada com o uso da extensão *relyBPMN* composta pelos elementos *ReliabilityInformation* e *Probability*. O elemento *ReliabilityInformation* é composto pelo valor mínimo de fiabilidade aceite num processo (*requiredReliability*) e pelo valor de fiabilidade calculado para um dado elemento BPMN (*calculatedReliability*) pelo algoritmo SWR. Já o elemento *Probability* tem um só valor e diz respeito à probabilidade de um elemento, pertencente a um bloco condicional ou bloco de ciclo, ser usado na execução.

### 2.2.4 Decomposição de processos BPMN

A grande maioria das redes *IoT* ainda apresenta uma arquitetura centralizada, onde um sistema central é responsável por executar toda a lógica do processo, e coordenar todos os aparelhos *IoT* envolvidos, tais como sensores e atuadores. Os sensores são usados na recolha de informação de um dado ambiente (temperatura e humidade, por exemplo), sendo estes dados transmitidos por mensagens para o sistema central. Este é responsável

por processar a informação e decidir o momento em que os atuadores realizam as respetivas funções.

No entanto, a comunicação entre o sistema central e os sensores gera um custo significativo, devido à quantidade de energia consumida através das trocas de mensagens a que são sujeitos. Além desse problema, também uma parte da segurança requerida em sistemas *IoT* pode-se revelar comprometida, no caso de os recursos disponíveis nesses aparelhos não serem suficientes no controlo de alguns dos possíveis riscos. Sensores destinados a *IoT* não foram desenhados em particular para suportarem controlos de segurança consideravelmente complexos, já que contêm uma energia limitada, e nem todos dispõem de carregadores de baterias. Três hipóteses poderiam ser consideradas para resolver essa limitação [29]. Uma seria minimizar os requisitos de segurança dos aparelhos, o que não é aconselhável devido à presença típica de dados sensíveis. Outra consistiria em aumentar a capacidade da bateria, o que seria um retrocesso no design esperado dos sensores, que apresentam a vantagem de serem leves e de pequenas dimensões. Uma terceira alternativa seria os sensores serem abastecidos com energia provinda de recursos naturais. Contudo, tal abordagem implicaria mudanças significativas ao nível do *hardware*, bem como um aumento no custo de produção.

Não havendo, por isso, uma solução completamente viável para o fornecimento de energia das baterias, é um facto que a realização de pequenas operações lógicas de decisão por parte dos sensores pode resultar num gasto consideravelmente inferior de energia, ao invés de dependerem totalmente da comunicação como meio para tomada de decisões no processo.

A solução proposta na literatura passa por se descentralizar o processo, de forma a que a comunicação entre as partes seja reduzida ao mínimo necessário, deslocando computação para os sensores. Para tal, os sensores podem também eles passar a fazer parte do processamento, tomando possíveis decisões que inicialmente caberiam à responsabilidade do sistema central. Ao nível do desenho do processo, isto requer mudanças das posições de vários elementos do processo, contudo mantendo-se as dependências de fluxos entre os elementos inalteradas.

Uma proposta para se efetuar a decomposição de processos BPMN é sugerida em [18], cuja ideia passa por detetar padrões formados por determinados elementos do processo. Consideremos um processo BPMN com duas *pools*, uma sendo a *pool* central, e a outra destinada aos sensores e atuadores. O objetivo passa por perceber se há cenários

com elementos na *pool* central a realizar processamento que possam ser transferidos para a *pool* de sensores, sem que a ordem de execução do processo se altere, bem como as dependências entre os elementos em questão. Caso tais cenários existam, estes podem ser decompostos, diminuindo o envio (assim desnecessário) de mensagens com pedidos e receções de informação.

No procedimento seguido começa-se por gerar um grafo com todos os fluxos de dados e de controlo do processo, e respetivas dependências. De seguida, é executado um algoritmo que percorre, a partir do nó inicial, os subcaminhos do grafo que contenham, pelo menos, uma mensagem trocada entre as *pools*, e cujos elementos façam parte da *pool* de sensores ou possam ser executados nessa *pool*. Subcaminhos que estejam contidos noutros caminhos são removidos, e os restantes são usados na reformulação das *pools*. Por fim, um novo modelo BPMN é gerado com a nova configuração. Em [10], a continuação da ideia anterior é desenvolvida, com novos padrões a serem detetados e usados como solução de decomposição do processo.

## 2.3 Casos de uso

De seguida, são descritos resumidamente dois casos de uso de processos de negócio dependentes da *IoT* que têm sido considerados no estudo de processos BPMN. O primeiro foi usado como exemplo na decomposição de processos [18]. O segundo caso foi estudado quanto à fiabilidade do sistema [25], com recurso ao método explorado em [24].

### 2.3.1 Sistema de irrigação

O primeiro caso de uso consiste de um sistema de irrigação automática, cuja água utilizada é armazenada previamente em tanques [18]. O sistema permite a leitura do nível da água presente no tanque, bem como do teor de humidade do solo, este último usado na decisão de quando se deve iniciar a irrigação. A Figura 2.2 apresenta o modelo BPMN com a representação do sistema.

O sistema é constituído por duas *pools*: uma é o sistema central e reflete o próprio processo de irrigação, sendo responsável por todo o processamento lógico; a outra – *pool IoT* – é composta pelos sensores (um para medir o nível da água no tanque, e outro para ler a humidade do solo) e atuadores (uma para iniciar a irrigação, e outro para reabastecer o tanque). O processo contém, por isso, dois fluxos de execução distintos: um para o reabastecimento do tanque; outro para a irrigação.

No fluxo de reabastecimento do tanque, o sistema central começa por pedir a um dos sensores o nível da água no tanque. O sensor lê e envia o valor para o sistema central, que calcula a quantidade de água necessária para encher o tanque. Essa quantidade é enviada então para o atuador, que inicia o reabastecimento, registando-se a ocorrência. Neste fluxo foram trocadas, no total, três mensagens.

No fluxo de irrigação, o sistema central pede ao sensor (por ação de um *trigger*) a leitura da humidade do solo. Recebido o valor, o sistema decide, com base neste, se se deve ou não irrigar. Se sim, o sistema notifica o atuador para iniciar a irrigação, e mais tarde (de acordo com um temporizador) para a concluir. O valor da humidade do solo é também persistido. No total, são trocadas quatro mensagens.

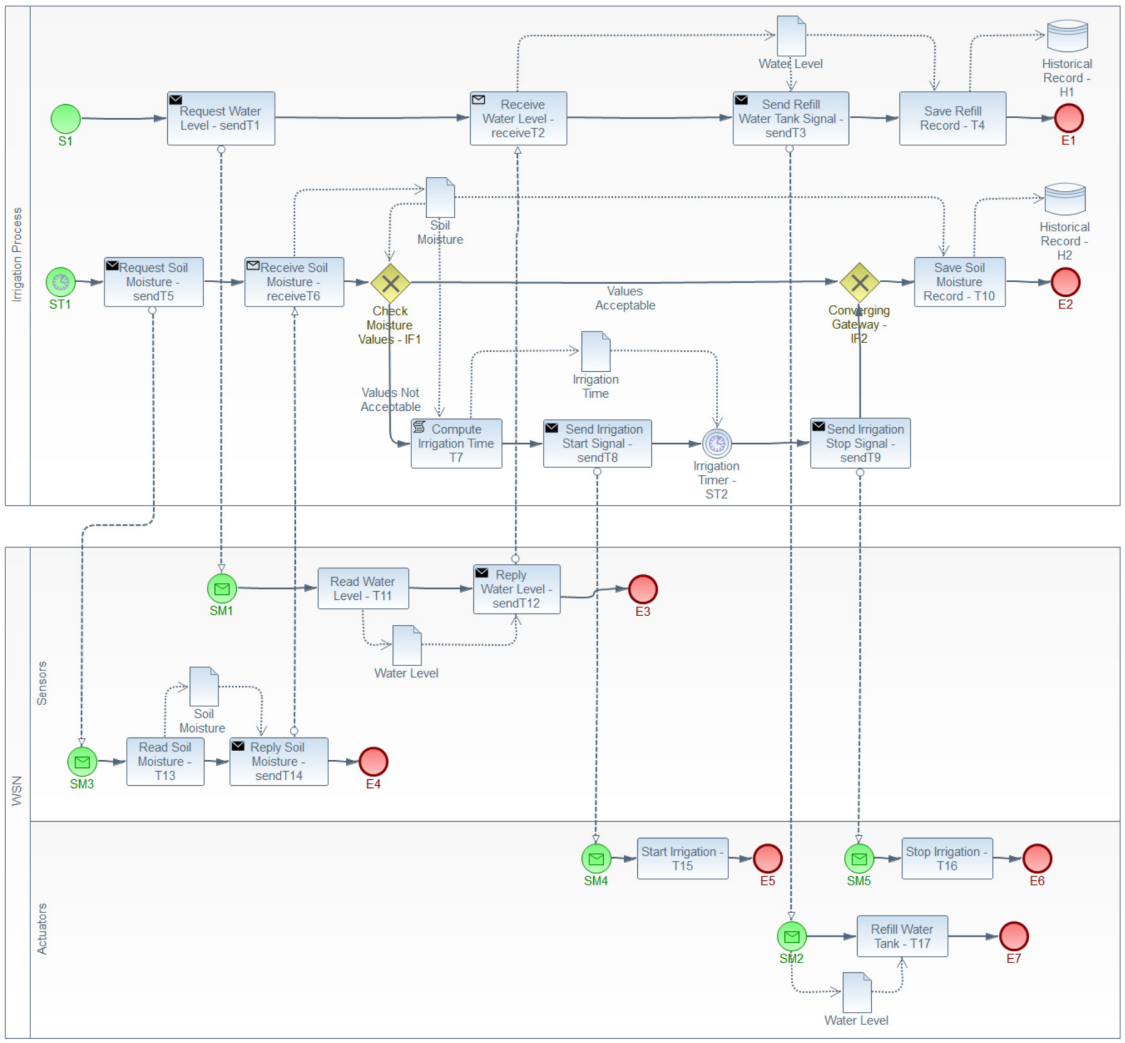


Figura 2.2 Modelo BPMN do Sistema de Irrigação Automática, extraído de [18]

O objetivo com este exemplo foi encontrar uma solução descentralizada, como referido em 2.1.5. Com a decomposição dos elementos do processo, parte da lógica de execução do processo foi transferida para a *pool* IoT, e um novo modelo BPMN foi gerado com um total de três mensagens trocadas – duas para o fluxo de reabastecimento, uma para o fluxo de irrigação.

### 2.3.2 Sistemas AAL (Ambient-Assisted Living)

Os sistemas AAL, usados no suporte a pacientes de risco, recolhem sinais vitais da pessoa periodicamente, e intervêm em caso de emergência. O objetivo deste tipo de sistemas é garantir que pacientes com necessidade constante de monitorização do seu estado de saúde conseguem manter consideravelmente a qualidade de vida, apesar das limitações. Um modelo BPMN de um sistema AAL é apresentado na Figura 2.3.

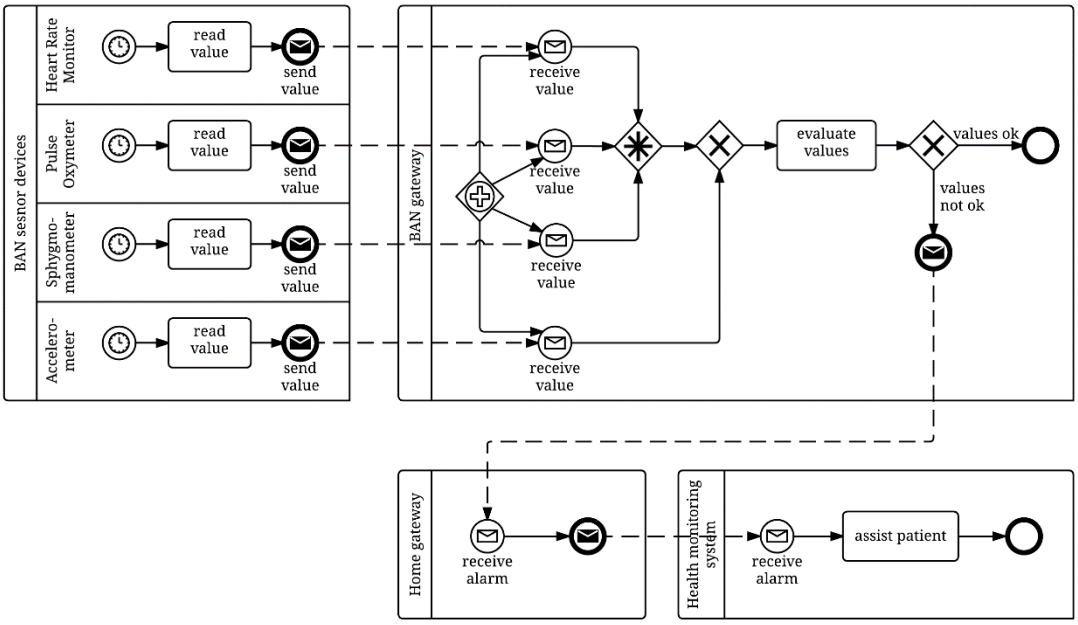


Figura 2.3: Modelo BPMN do Sistema AAL, extraído de [25]

O sistema ALL exemplificado em [25] é composto por quatro *pools* distintas. A primeira *pool* – aparelhos da *Body Area Network* (BAN) – tem quatro sensores: três deles monitorizam atividade cardíaca registando a frequência cardíaca, saturação de oxigénio e pressão sanguínea; o outro trata-se de um acelerómetro destinado a monitorizar a atividade corporal.

A segunda *pool*, BAN *gateway*, recebe e avalia os valores lidos pelos sensores. Se o paciente se encontrar em estado de emergência um alarme é enviado, por intermédio da *Home gateway* (terceira *pool*, que pode ser uma peça de *hardware*, como um *router*, por exemplo), até ao HMS (Sistema de Monitorização de Saúde), notificando o responsável pela condição do paciente sobre a corrente situação.

Foi imposto inicialmente que o valor mínimo aceite para a fiabilidade do sistema seria 0.6 (de 0 a 1). Dado um conjunto inicial de parâmetros, o cálculo da fiabilidade geral do processo foi efetuado através do algoritmo SWR, referido em 2.1.3, e resultou no valor 0.6901, significando que esta configuração do sistema se encontra acima do mínimo praticável. Foram ainda executados vários testes para perceber o impacto individual e combinado de vários elementos na fiabilidade do sistema. Para tal, foram feitas alterações aos valores de fiabilidade ao nível dos sensores (tanto individualmente como aos pares), da transmissão destes para a BAN *gateway*, e da transmissão da BAN *gateway* para o HMS.

## 2.4 Otimização combinatória multi-objetivo

A otimização combinatória é o método que procura obter, de entre um conjunto finito de soluções, aquela(s) que melhor corresponde(m) a um determinado critério de escolha, designado de objetivo [3]. Usado na melhoria de vários setores, este tipo de otimização tem sido amplamente estudado pela ciência da computação.

Um problema de otimização multi-objetivo pode ser definido como o intuito de [22]: encontrar um vetor de valores para as variáveis de decisão que satisfaça um conjunto de restrições e otimize uma função vetorial cujos elementos representam as funções objetivo. Estas funções formam uma descrição matemática de critérios de desempenho que tipicamente entram em conflito entre eles. Assim, o termo "otimizar" significa encontrar uma solução que forneça valores para todas as funções objetivo aceitáveis ao designer.

Formalmente, podemos afirmar [5]:

Pretende-se encontrar um vetor $\bar{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]^T$ que satisfaça as $m$ restrições de desigualdade:

$$g_i(\bar{x}) \geq 0 \quad i = 1,2,\ldots,m \tag{1}$$

as *p* restrições de igualdade

$$h_i(\bar{x}) = 0 \quad i = 1,2,\ldots,p \tag{2}$$

e otimize a função vetorial

$$\bar{f}\,(\bar{x}) = [f_1(\bar{x}), f_2(\bar{x}), \ldots, f_k(\bar{x})]^{\,T} \tag{3}$$

onde $\bar{x} = [x_1, x_2, \ldots, x_n]^T$ é o vetor de variáveis de decisão.

Isto é, pretendemos encontrar de entre o conjunto de todos os vetores que satisfazem (1) e (2) o conjunto específico $x_1^*, x_2^*, \ldots, x_k^*$ que produz os valores ótimos para todas as funções objetivos.

Um vetor de variáveis de decisão, que satisfaça as *m* restrições de desigualdade e *p* restrições de igualdade impostas, é por isso considerado uma solução viável/admissível. O conjunto total de soluções viáveis do problema é denominado por espaço de soluções.

### 2.4.1 Definições em otimização multi-objetivo

Na otimização multi-objetivo, a solução do problema é um conjunto que pode ser constituído por uma ou mais soluções viáveis, designado por conjunto de soluções não dominadas, ou conjunto de Pareto.

*Dominância*:

Para um problema de maximização, uma solução *a* domina uma solução *b* se:

(1) *a* não tem nenhum objetivo cujo valor seja inferior ao valor desse objetivo em *b* e

(2) *a* tem, pelo menos, um objetivo cujo valor é superior ao valor desse objetivo em *b*.

Se uma solução não for dominada por nenhuma outra, então é uma solução não dominada, e pertencerá ao conjunto de Pareto. Caso contrário, é uma solução dominada.

S*olução de um problema de otimização multi-objetivo*:

A solução de um problema de otimização multi-objetivo é o conjunto das suas soluções admissíveis não dominadas.

*Solução equivalente*:

Duas soluções *a* e *b* de um problema multi-objetivo dizem-se equivalentes sse *a* não domina *b* nem é dominada por *b*.

*Conjunto/fronteira de Pareto*:

O conjunto/fronteira de Pareto é a representação das soluções não dominadas no espaço dos objetivos. Uma solução não dominada corresponde a uma solução Pareto eficiente (ótima). Uma solução dominada corresponde a uma solução Pareto não-eficiente.

A Figura 2.4 representa um conjunto de Pareto para um problema bi-objetivo de maximização, dados dois objetivos, Objetivo 1 e Objetivo 2. As soluções A, B e C são soluções eficientes, ou seja, encontram-se na fronteira de Pareto, contrariamente a D, que se trata de uma solução dominada.

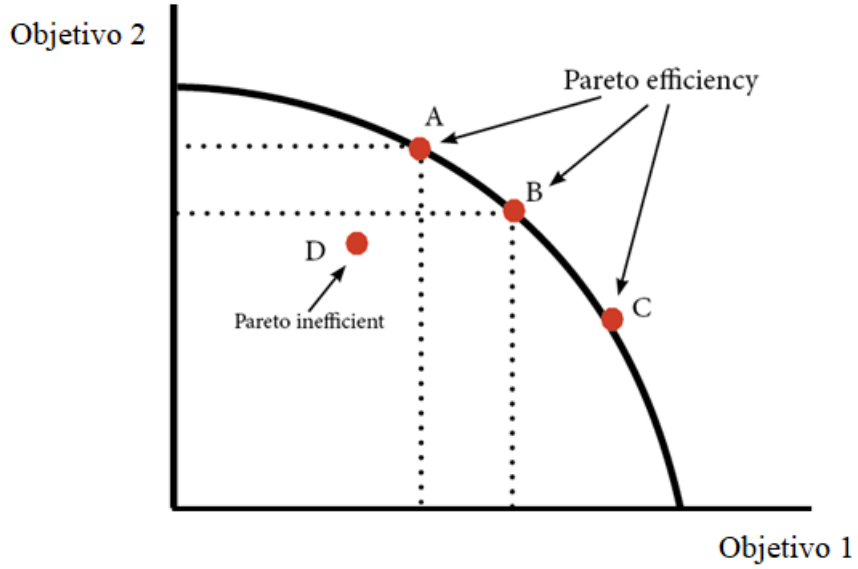


Figura 2.4 Conjunto de Pareto num problema bi-objetivo, retirado de [23]

## 2.5 Algoritmos meta-heurísticos

A classe de algoritmos meta-heurísticos foi criada com o propósito de diminuir o tempo de computação resultante do uso da busca exaustiva, nomeadamente em problemas de otimização com um espaço de soluções tão vasto, que o tempo de espera tornar-se-ia incomportável. Estes são algoritmos com os quais se conseguem resultados próximos dos

ótimos, e são também utilizados em problemas para os quais não existe uma formulação matemática que os relacione com um problema para o qual existe um algoritmo exato. À falta de uma definição formal para meta-heurísticas, estas consistem de processos iterativos que procuram determinar de forma eficiente as soluções aproximadamente ótimas contidas num espaço de procura [3]. Atuam geralmente de maneira não-determinística devido ao uso de componentes probabilísticas, para ultrapassarem fragilidades encontradas em alguns dos típicos algoritmos determinísticos, tais como a tendência para ficarem retidos em extremos locais, ao invés de descobrirem o desejado extremo global, ou de dependerem de um tempo de computação demasiado elevado. Não sendo garantido que encontrem a solução ótima para um dado problema (ou conjunto ótimo, se houver mais de um objetivo), estas conseguem, ainda assim, uma boa aproximação a essa solução. Meta-heurísticas podem ser vistas como métodos, que mantêm um nível de abstração que lhes permite serem aplicados a diversos problemas de otimização. A escolha do método mais adequado nem sempre é intuitiva, pois depende das características do problema em causa, nomeadamente o seu número de objetivos.

De entre as várias meta-heurísticas criadas ao longo do tempo, destacam-se *Ant Colony Optimization* (ACO), *Simulated Annealing* (SA), *Tabu Search* (TS) e algoritmos de computação evolutiva tais como *Nondominated Sorting Genetic Algorithm* (NSGA) e *Strength Pareto Evolutionary Algorithm* (SPEA), além de outras alternativas terem sido criadas a partir dessas. Como tal, nem sempre a classificação de um dado algoritmo é simples e direta. Devido à natureza probabilística das meta-heurísticas referidas, designamos este tipo de otimização por Otimização Combinatória Estocástica [2]. De seguida, é descrito o comportamento genérico de métodos de busca local (*Local Search*), bem como o do algoritmo *Simulated Annealing*.

### 2.5.1 Pesquisa Local

O algoritmo de Pesquisa Local (*Local Search*) é um dos algoritmos de otimização mais simples e baseia-se no conceito de tentativa e erro. O seu comportamento geral, como descrito na Figura 2.5 para um problema de maximização, é o seguinte: uma solução inicial é gerada, e é definida como a solução atual. De seguida, o espaço das soluções vizinhas da atual solução é percorrido até que uma vizinha melhor seja encontrada. Se tal acontecer, a vizinha passa a ser a nova solução atual, e a procura prossegue nas soluções da sua vizinhança. A execução termina quando nenhuma solução vizinha da atual for

melhor que esta última. A solução atual que se mantiver no final da execução é declarada como a melhor solução encontrada.

```
INÍCIO
Ler função de custo f e a estrutura de vizinhança N
Selecionar uma solução s0
Enquanto f(s) < f(s0) para alguma s ∈N(s0) Fazer
        Início
                Selecionar s ∈ N(s0) tal que f(s) < f(s0);
                s0: = s;
        Fim
s0 é uma aproximação da solução ótima
FIM
```

Figura 2.5 Pseudo-código do algoritmo *Local Search* [12]

A principal desvantagem de algoritmos como o *Local Search* é a possibilidade de a procura se fixar num extremo local, sem conseguir sair de lá. Isto acontece quando na vizinhança da solução atual não existe nenhuma alternativa melhor, impedindo o algoritmo de explorar o restante espaço de soluções. Nem sempre esta ocorrência determina um mau resultado – pode acontecer que o valor da função objetivo do extremo local esteja pouco abaixo do valor do extremo global. Contudo, é um acontecimento pouco provável de ocorrer, à medida que o espaço de soluções aumenta. No melhor dos casos, o espaço de soluções teria um formato convexo, e o *Local Search* seria suficiente para encontrar o máximo global devido à ausência de extremos locais.

### 2.5.2 Simulated Annealing

O *Simulated Annealing* (SA) foi introduzido em [14] e endereçava o problema do caixeiro viajante (*TSP*), que consiste em, para uma lista de N cidades e com todas as distâncias conhecidas entre cada duas delas, calcular a viagem de menor custo a começar e a terminar na mesma cidade, passando por todas elas uma só vez. Encontrar a solução ótima do problema por meio de busca exaustiva seria viável para um pequeno número N. À medida que o número de cidades aumenta, o espaço de soluções cresce radicalmente, tornando esse método inadequado, devido ao tempo de execução requerido. Ainda assim, o *simulated annealing* revelou-se um algoritmo com boas aproximações quanto à melhor rota – com 310 cidades ainda foi possível encontrar a melhor solução, dos estudos conhecidos até à altura. Devido à sua boa performance (conforme a escolha apropriada dos parâmetros), o algoritmo tem sido extensivamente estudado e readaptado conforme as características dos problemas de otimização que vão aparecendo.

Este algoritmo, inspirado na natureza, baseia-se no arrefecimento moderado de metais, começando tipicamente com uma temperatura inicial elevada. Cada iteração equivale a uma diminuição no valor de temperatura do metal, de acordo com uma dada taxa de arrefecimento que, por norma, é pequena. A execução termina quando a temperatura atinge um valor mínimo inicialmente estabelecido. A principal diferença do *Simulated Annealing* para algoritmos como o *Local Search* está na forma como a solução atual é mantida: uma solução vizinha da solução atual pode até ser pior que esta última e, ainda assim, ser aceite como nova solução atual. A decisão recai numa probabilidade de aceitação, que é calculada com base no estado corrente da execução. É esta possibilidade de ocorrer uma mudança para soluções piores que faz com que a procura não termine inevitavelmente presa num extremo local.

A probabilidade de aceitação é calculada com base na variação das energias (valores da função objetivo) entre as duas soluções comparadas numa dada iteração. No entanto, para assegurar que o algoritmo vai iterativamente aproximando-se da solução ótima, também a temperatura atual é usada como parâmetro no cálculo da probabilidade. Devido a isso, torna-se importante uma escolha conveniente dos parâmetros de temperatura inicial e taxa de arrefecimento, bem como da fórmula usada no cálculo da probabilidade de aceitação, pois esta influencia a decisão de ocorrer, ou não, uma transição de estado. No entanto, a atribuição de valores a estes parâmetros não é simples e direta, porque dependem muito do problema em estudo. Por isso, a escolha acertada dos parâmetros recai frequentemente em análises de tentativa e erro.

O funcionamento do *Simulated Annealing*, bem como a respetiva fórmula de cálculo para a probabilidade de aceitação, são baseados no algoritmo de Metropolis, introduzido em 1953 e destinado a resolver problemas de mecânica estatística, com recurso ao método Monte Carlo, procedimento usado na geração de amostragens aleatórias [19]. A probabilidade de ocorrer a mudança de um estado *S* para um estado *S'*, num problema de maximização, é representada por:

$$P(Accept\ S') = min\left(1, \exp\left\{\frac{\Delta E}{K \times T}\right\}\right) \quad (4)$$

onde:

- $\Delta E = E(S') - E(S)$. $\Delta E$ representa a variação de energia resultante de ocorrer a mudança de estado. $\Delta E > 0$ significa que o valor da função objetivo associada ao estado S', *E(S')*, é superior ao da função objetivo do estado S, *E(S)*. $\Delta E = 0$ significa que não houve variação. Se $\Delta E < 0$, houve uma

diminuição da energia do sistema, ou seja, a nova solução com estado S' tem um valor de função objetivo inferior ao correspondente do estado S.

- *K* é a constante de Boltzman e, por norma, é-lhe atribuído valor igual a 1.
- *T* é a temperatura atual do sistema.
- $\exp\left\{\frac{\Delta E}{K \times T}\right\}$ é denominado de fator de Boltzman, Para qualquer ΔE > 0, o fator $\frac{\Delta E}{K \times T}$ será também positivo, logo $P(Accept\ S')$ será sempre igual a 1. Pelo contrário, se a variação de energia for negativa, então $\frac{\Delta E}{KT} < 0$. Neste caso, $P(Accept\ S')$ será menor que 1.

Posteriormente, versões do SA foram sendo adaptadas para problemas com dois ou mais objetivos. Os algoritmos MOSA (*Multi-objective Simulated Annealing*), como muitos outros, mantêm uma lista de soluções não dominadas durante toda a procura, e as que restarem nesse conjunto, terminada a execução, traduzem-se na fronteira de Pareto. Em cada iteração é criada uma mutação da solução atual, e são comparadas as duas. Se a mutação não for dominada pela atual, a lista mantida é corrida e as soluções dominadas pela mutação são removidas. A mutação é adicionada ao conjunto se não for dominada por nenhuma das outras na lista.

O primeiro algoritmo MOSA foi sugerido por Serafini [26], onde um conjunto de hipóteses para o cálculo da probabilidade de aceitação são sugeridas. As probabilidades são distinguidas em dois critérios: forte e fraco. No critério forte, a transição para um novo estado apenas é garantida de ocorrer se a nova solução for melhor. No critério fraco, basta que nova solução seja não dominada para que ocorra uma transição de estado com probabilidade 1. Novas adaptações do MOSA começaram subsequentemente a ser criadas.

Neste estudo, é destacado o algoritmo SMOSA (*Suppapitnarm Multiobjective Simulated Annealing*), desenvolvido em [28], e que mostrou bons resultados para problemas de otimização investigados no artigo, com dois e três objetivos. Neste caso, a probabilidade de ocorrer uma transição consiste no produto de diferentes probabilidades de aceitação calculadas para cada um dos objetivos, onde é seguido o critério forte. Para um problema de maximização, a probabilidade de ocorrer a transição de um estado *S* para *S'* é a seguinte:

$$P(Accept\ S') = min\left(1, \prod_{i=1}^{M} exp\left\{\frac{\Delta E_i}{T_i}\right\}\right) \quad (5)$$

onde M é o número de objetivos do problema, $\Delta E_i$ e $T_i$ a variação de energia e temperatura, respetivamente, para um objetivo *i*. Ou seja, existem diferentes variáveis de temperatura para cada um dos objetivo. O uso de diferentes temperaturas recai no facto de, para uma dada execução, a média das variações de energia para um objetivo poder diferir bastante da média das variações dos outros objetivos. Usar a mesma temperatura para todos eles faria com que a procura tendesse mais no sentido de um objetivos do que dos outros. O que, tipicamente, se pretende num problema de otimização é efetuar a procura favorecendo a melhoria, não de um objetivo em particular, mas de todos eles em simultâneo. Assim, utilizar diferentes variáveis de temperaturas adequadas a cada objetivo asseguram a imparcialidade da busca. Esta abordagem é também vantajosa na medida em que as temperaturas são calculadas com uma base de conhecimento do problema, ao invés de se usar um método de tentativa e erro.

No início da execução, todas as transições de estado são aceites, ou seja, a probabilidade de aceitação é sempre igual a 1 (as temperaturas podem ser assumidas como infinitas). Por cada objetivo existente, é mantido um conjunto que regista o valor desse objetivo para a solução aceite. Sempre que uma transição de estado ocorre, os valores da nova solução são adicionados aos respetivos conjuntos. Ao fim de um número estipulado de iterações $N_{T1}$, é calculado o desvio padrão para os valores de cada um dos conjuntos. O resultado de cada desvio padrão determina a temperatura inicial para cada objetivo. A partir daí, a transição de estado para uma nova solução pior passa, efetivamente, a ter probabilidade de aceitação inferior a 1. As temperaturas passam a ser reduzidas a cada $N_{T2}$ iterações ou $N_A$ transições de estado aceites, até que a execução do algoritmo termine. A taxa de arrefecimento para cada objetivo é também determinada com conhecimento do problema, recorrendo-se aos valores atuais de temperatura e desvio padrão do respetivo conjunto.

Também uma estratégia, denominada de retorno à base, é utilizada periodicamente. Esta estratégia impõe que, ao fim de $N_{Bi}$ iterações, a busca é recomeçada numa solução contida no atual conjunto não dominado. O estudo recomenda que, para um problema com *M* objetivos, a escolha seja efetuada de entre as *M* soluções nos extremos do conjunto. Isto é, deve-se considerar as soluções que tenham o valor mais

elevado (assumindo um problema de maximização) para cada um dos $M$ objetivos. É, ainda, sugerido que se adicione às possíveis escolhas a solução mais distante relativamente a todas as outras (sem contar com as $M$ soluções dos extremos). A frequência do retorno à base vai aumentando ao longo da execução, ao se diminuir o valor $N_{Bi}$ após cada uso da estratégia. O algoritmo é concluído após um número total de iterações (decidido previamente) ser alcançado.

Além do SMOSA, outros métodos baseados no SA apresentaram resultados satisfatórios, tais como UMOSA, PSA, WDMOSA e PDMOSA [27]. O PSA, por exemplo, combina o *simulated annealing* com o conceito de população, presente na classe dos algoritmos genéticos. Uma população é um conjunto de soluções geradas aleatoriamente no início da execução de um algoritmo que se faz evoluir, adicionando os elementos não dominados ao conjunto mantido durante toda a execução.

### 2.5.3 Condição de paragem

Existem vários critérios de paragem da execução das meta-heurísticas. A condição mais utilizada será, muito possivelmente, após o algoritmo chegar a um número de iterações estipulado ao início. É a condição mais fácil de implementar, pois não depende do problema em causa, ou seja, ignora as variáveis da análise. Outras condições, pelo contrário, baseiam-se na informação recolhida durante a execução como auxílio na tomada de decisão. [11] enumera as principais condições de paragem e agrupa-as em cinco classes, além de sugerir uma classe de combinações entre elas:

1. Critério de referência (ou baseado em erro): usado quando o conjunto ótimo conhecido à partida, o que raramente ocorre em situações do mundo real, onde o espaço de soluções é tipicamente vasto. O algoritmo termina ao atingir uma dada proximidade à solução ótima.
2. Critério baseado na exaustão: quando o algoritmo atinge um número estipulado de iterações ou gerações (termo usado nos algoritmos evolutivos), avaliações de função objetivo ou tempo de CPU.
3. Critério baseado na melhoria: se ocorrem poucas ou nenhumas melhorias significativas na procura de soluções.
4. Critério baseado no movimento: foca-se no movimento entre soluções. Requer o rastreamento de várias soluções, e é especialmente útil em algoritmos evolutivos, devido ao uso de populações.

5. Critério baseado na distribuição: também usado em métodos baseados em populações, assume que o algoritmo convergiu após os elementos se concentrarem numa pequena área.

### 2.5.4 Comparação de conjuntos não-dominados

Na otimização de problemas com um só objetivo, o critério de procura da melhor solução é simples de pressupor – será aquela que apresente o maior valor calculado pela função objetivo, quando num problema de maximização (ou menor valor, se o problema for de minimização). Em problemas multi-objetivo, a complexidade da procura aumenta, devido à seguinte situação: uma solução *a* pode ter um valor superior à de uma solução *b*, para um objetivo *O1*, mas ter um valor inferior à de *b* para um objetivo *O2*, ou seja, nem *a* domina *b*, nem *b* domina *a*. Neste caso, não sendo possível determinar diretamente qual das duas se trata da melhor solução, teremos não uma (e única) solução ótima, mas sim um conjunto de soluções ótimas, que não se dominam entre elas.

De forma a avaliar os diferentes conjuntos de soluções não dominadas, recorremos a indicadores de performance dos algoritmos, os quais designamos de *métricas*. Esta avaliação de dois ou mais objetivos revela-se, contudo, uma tarefa consideravelmente mais árdua. Primeiro, porque os algoritmos usados podem nem sempre produzir conjuntos de soluções de qualidade aceitável. Segundo, porque não existe um critério de comparação que seja considerado o melhor, de forma unânime, para todas as situações Como tal, é importante que os conjuntos de soluções geradas pelos algoritmos tenham em conta, e sempre que possível, as seguintes características [31]:

- A distância entre o conjunto gerado de soluções não dominadas e a fronteira de Pareto pretendida deve ser minimizada.
- É desejável uma boa distribuição das soluções encontradas, com base numa determinada métrica escolhida.
- A extensão da frente não dominada obtida deve ser maximizada, ou seja, para cada objetivo, deve haver uma vasta amplitude de valores produzidos.

Ao longo dos anos, vários trabalhos foram realizados quanto ao desenvolvimento de métricas, úteis na comparação dos algoritmos de otimização multi-objetivo bem como dos conjuntos de soluções não dominadas. A par disso, também foram efetuadas análises com vista a escolher a métrica mais adequada conforme as características do problema

(por exemplo, o número de objetivos). De entre os indicadores de qualidade revistos, o *hypervolume* de Zitzler (ou métrica S) [33] destacou-se como um dos mais consensuais, e foi também escolhido para usar nesta investigação. O trabalho [15] recomenda o uso do *hypervolume* para problemas de otimização com relativamente poucas dimensões e cujos conjuntos não-dominados não sejam demasiado extensos, de modo a evitar sobrecarga computacional durante a execução.

O *hypervolume* mede o tamanho do espaço delimitado pelo conjunto de pontos não-dominados e por um ponto de referência à escolha. Como condição, as soluções devem-se encontrar ordenadas, e o mesmo ponto de referência deve ser usado para todos os conjuntos a serem comparados.

# Capítulo 3

# Otimização da descentralização da computação em processos de negócio IoT

Este capítulo apresenta o trabalho desenvolvido. É definido o problema de otimização e a sua finalidade. São descritos os algoritmos desenvolvidos, um de busca exaustiva e dois meta-heurísticos. Ao longo deste capítulo, o processo de irrigação automática, apresentado em 2.3.1, será utilizado como processo ilustrativo na explicação de alguns dos passos do trabalho.

## 3.1 Definição do problema

Os processos de negócios são fundamentalmente constituídos por objetos de fluxo, participantes e ligações. Os objetos de fluxo no BPMN, podem consistir de atividades (tarefas ou subprocessos), eventos ou *gateways*. Os participantes designam-se de *pools* no BPMN, e definem a posição de todos os objetos de fluxo do processo. Isto é, a execução de um objeto de fluxo fica sempre à responsabilidade de um dos participantes. Uma *pool* pode ainda conter *lanes*, que representam sub-entidades do participante em questão. Por exemplo, é comum acontecer que uma *pool* destinada aos aparelhos *IoT* tenha, pelo menos, uma *lane* destinada aos sensores e outra *lane* para os atuadores.

As ligações (objetos de ligação no BPMN) do processo determinam a ordem de execução dos objetos de fluxo. Dois objetos de fluxo no mesmo participante são ligados através de um fluxo de sequência. Se estiverem situados em diferentes participantes, então serão ligados por meio de um fluxo de mensagem. As ligações podem também consistir de associações de dados, que representam o fluxo de informação utilizada ao longo do processo. Essa informação pode consistir de *data objects* (existem vários no processo de irrigação, tais como os valores lidos pelos sensores), pode ser *data stores*, usados para guardar vários registos (o processo de irrigação tem dois, *Historical Record - H1* e *Historical Record - H2*), e pode ainda ser *inputs* usados pelo processo ou *outputs* por este produzido.

Daqui para a frente, por simplicidade, quando for mencionado o termo *elemento(s)*, estamos a referir-nos especificamente a objeto(s) de fluxo. Para se efetuar a descentralização dos processos, é necessário separar os elementos por dois grupos: os que podem mudar de participante (elementos mutáveis) e os que não podem (elementos fixos). A realização de uma pequena operação lógica pelo sistema central é, muito possivelmente, um exemplo de elemento mutável, pois esta pode ser transferida para a *pool IoT*, e passar a ser uma das tarefas realizadas por um sensor. Pelo contrário, medir o nível da água no tanque (processo de irrigação) é uma tarefa que só faz sentido de ser executada pela *pool IoT*, logo tratar-se-ia de um elemento fixo. Qualquer um dos elementos do tipo tarefa pode assumir apenas uma das funções seguintes:

- As que executam uma atividade específica do processo, tais como as operações lógicas e leitura de valores, acimas referidos. Dependendo da natureza do processo, estas tarefas podem tanto ser elementos mutáveis como fixos.
- As que realizam a comunicação entre diferentes participantes, ou seja, que são responsáveis pelo envio de mensagens (por norma, entre o sistema central de execução e os aparelhos *IoT*). Por cada comunicação ocorrida no processo, haverá sempre uma tarefa de envio e uma tarefa de receção da mensagem, situadas em diferentes participantes. Nenhuma destas tarefas é, em caso algum, um elemento mutável, já que apenas são utilizadas para que um participante requeira a realização de uma atividade por parte de outro participante. É este tipo de tarefas que determina o número de comunicações que existem num processo, e consequentemente, o respetivo custo total de se efetuar comunicação.

Dada a descentralização dos processos de negócio dependentes de *IoT*, este problema de otimização procura encontrar soluções alternativas do processo original, onde o número de tarefas de envio e receção de mensagens entre participantes seja o mais pequeno possível. Isso significará um menor número de comunicações, e posteriormente, um menor consumo de energia por parte dos aparelhos *IoT*, resultando numa redução do custo. A diferença fundamental entre todas as soluções está no participante responsável por executar cada um dos elementos mutáveis. Basta haver a variação de participante de apenas um elemento mutável, para essa constituir uma nova solução do processo. Os elementos fixos do processo não permitem distinguir as diferentes soluções, porque o participante responsável por cada um desses será sempre o mesmo, independentemente

da solução. Por outro lado, também o valor de fiabilidade total de uma nova solução do processo pode ser diferente do valor da do modelo original, tanto para melhor como para pior. Uma solução será, assim, avaliada através de um par de valores custo de comunicação e fiabilidade.

O problema de otimização procura obter a solução que apresenta o melhor par de valores que minimiza o custo e maximiza a fiabilidade. Uma vez que estamos a lidar com dois objetivos em conflito, eventualmente haverá, não uma solução ótima, mas um conjunto de várias soluções que apresentem uma fiabilidade elevada e, ao mesmo tempo, um custo de comunicação o mais pequeno possível, devido ao *trade-off* entre os objetivos (uma redução no custo de comunicação pode levar a uma redução no valor da fiabilidade).

## Funções objetivo

São consideradas as duas funções objetivo seguintes para o problema de otimização: Maximização da fiabilidade geral do processo e Minimização do custo de comunicação do processo.

- **Maximização da fiabilidade geral do processo**

A fiabilidade geral do processo é determinada através dos valores de fiabilidade de cada tarefa nele contida. O método utilizado é o *Stochastic Workflow Reduction* proposto em [6], que identifica um conjunto de padrões de seis tipos possíveis num *workflow* (processo), designados de blocos de processo. Cada bloco é constituído por duas ou mais tarefas, sendo-lhe aplicada uma redução até ser representado por uma só tarefa. A fiabilidade dessa tarefa será também a fiabilidade do bloco. O método é realizado iterativamente até que, no final da execução, reste apenas uma tarefa representativa de todo o *workflow*. O valor de fiabilidade da última e única tarefa expressa o valor de fiabilidade geral do processo, e encontra-se no intervalo [0,1].

- **Minimização do custo de comunicação do processo**

O custo de comunicação do processo representa a soma dos custos provenientes das comunicações realizadas entre diferentes participantes do processo. Assumindo que o custo de um dado processo é determinado a partir de um diagrama D que o representa (e onde *pool* é o termo em BPMN para participante), o custo de comunicação de D, *cost(D)*, é o seguinte:

$$cost(D) = \sum_{(i,j)\in D} cost\big(pool(i), pool(j)\big) \quad (6)$$

onde *i* é o elemento emissor da comunicação e *j* o elemento recetor, *pool(n)* é a *pool* onde se situa o elemento *n*, e *cost(a, b)* é o custo da transmissão a partir da *pool a* para a *pool b*. Este valor *cost(D)* pode tomar valores entre zero e um valor positivo, e tipicamente é bastante superior à unidade, devido ao custo de comunicação entre cada duas *pools* ser aqui considerado um valor inteiro positivo. É também assumido, neste trabalho, que o custo de comunicação é o mesmo para todos os elementos, conforme os participantes emissor e recetor em questão.

O custo é ainda normalizado, para se encontrar no intervalo [0,1]. Isto permite que os dois eixos do quadrante, onde é definido o plano dos objetivos, tenham o mesmo intervalo de valores. Para o cálculo desse valor, utilizamos como denominador o valor máximo possível para o custo de comunicações no processo representado pelo diagrama *D*, $COE_{max}(D)$, de acordo com a Equação (7):

$$COE_{max}(D) = \sum_{(i,j)\in D} \max\left(cost\left(pool(i), pool(j)\right)\right) \quad (7)$$

O valor do custo normalizado, *costNormalized*(*D*), é então calculado através da Equação (8):

$$costNormalized(D) = \frac{cost(D)}{COE_{max}(D)} \quad (8)$$

Para melhorar a eficácia das metodologias a usar na procura de soluções, optou-se por otimizar em alternativa uma outra função bi-objetivo, com os dois objetivos a variar no mesmo intervalo [0;1] e ambos de maximização. Para tal, a minimização do custo de comunicações é substituída pela maximização da contraparte do custo de comunicações normalizado. Minimizar $costNormalized(D)$ é equivalente a minimizar $cost(D)$. Como $costNormalized(D)$ pertence a [0;1], minimizar esse valor é equivalente a maximizar a sua contraparte, ou seja, $1 - costNormalized(D)$. Deste modo, ambos os objetivos passam a ter os seus valores compreendidos no intervalo [0,1], e a natureza do problema passa a ser de dupla maximização, onde se procura:

- Objetivo 1: Maximizar a fiabilidade do processo e
- Objetivo 2: Maximizar a contraparte do custo de comunicação

A título de exemplo, calculemos o par de valores das funções objetivos da solução correspondente ao modelo original do processo de irrigação automática (ou seja, sem ter ocorrido a descentralização), assumindo que este é traduzido por um diagrama D.

A fiabilidade determinou-se através do método, já referido, de identificação e redução de blocos encontrados no diagrama. O valor resultante foi de aproximadamente 0.48182 para o objetivo 1.

Para o cálculo da contraparte do custo, começámos por considerar valores para os custos de comunicação entre as *pools* 1 e 2, tal que *cost*(*1,2*) = 7 e *cost(2,1)* = 13. Dos 28 elementos que constituem o processo, 16 situam-se na *pool* 1, e os restantes 12 na *pool* 2. Para calcular o custo máximo de comunicação $COE_{max}(D)$, devemos assumir que todos os elementos na *pool* 1 realizam uma comunicação para a *pool* 2, e vice-versa. Se para um elemento na *pool* 1, *cost*(*1,2*) = 7, então para 16 elementos, o custo total será 16 × *cost*(*1,2*) = 112. No sentido inverso da comunicação, se para um elemento na *pool* 2, *cost(2,1)* = 13, o custo total para os 12 elementos será 12 × *cost(2,1)* = 156. Logo, o custo máximo de comunicação será a soma desses dois valores: $COE_{max}(D)$ = 112 + 156 = 268.

O diagrama D contém sete comunicações entre *pools*, das quais cinco são enviadas por elementos da *pool* 1, e duas enviadas a partir da *pool* 2. O custo da comunicação, *cost(D)*, é calculado da mesma forma que em $COE_{max}(D)$, mas apenas para o número de comunicações que são efetivamente realizadas. Ou seja, *cost(D)* = 5 × *cost(1,2)* + 2 × *cost(2,1)* = 5 × 7 + 2 × 13 = 61. O valor normalizado desse custo traduz-se em 61/268.

Como pretendemos a contraparte do custo normalizado, $1 - costNormalized(D)$, então teremos 1 – (61/268) ≈ 0.77239 como valor final da função objetivo 2.

A solução correspondente ao modelo original do processo de irrigação tem, assim, associado o par de valores (0.48182, 0.77239) no espaço dos objetivos para fiabilidade e contraparte do custo, respetivamente.

## 3.2 Complexidade do processo

O espaço de soluções de cada instância deste problema (um processo) é constituído por todas as combinações diferentes de participantes onde os elementos mutáveis podem ser executados. Para se determinar a dimensão do espaço de soluções, é suficiente saber por quantos participantes cada um dos elementos mutáveis pode ser executado. No contexto

do problema de otimização em curso, a dimensão do espaço de soluções de um processo é designada de complexidade do processo.

Consideremos, por exemplo, um processo com apenas dois participantes e cinco elementos, dos quais apenas três são mutáveis. A complexidade do processo, dada a sua descentralização, é igual a $1\times1\times2\times2\times2 = 8$. Ou seja, o processo terá, no total, oito soluções possíveis. De realçar que num processo com, por exemplo, três participantes, nem todos os elementos mutáveis têm necessariamente de ser executáveis pelos três participantes – alguns podem ter apenas dois participantes como hipóteses possíveis.

O valor de complexidade revelou-se a métrica mais apropriada para a caracterização dos processos. A dimensão de um processo (número total de elementos) podia ser uma das alternativas consideradas como método de classificação. Contudo, correr-se-ia o risco de essa revelar resultados poucos conclusivos para o nosso propósito de otimização. Embora um processo P1 possa ter maior dimensão que um processo P2, isso não garante que a sua complexidade também seja superior à de P2. A complexidade é, em suma, uma representação do número de participantes onde cada um dos elementos mutáveis se pode situar, e são esses dois valores os responsáveis pela maior parcela do tempo investido na procura das soluções. A execução dos algoritmos em P2 poderia levar mais tempo que a execução em P1 e, como tal, classificá-los quanto à sua dimensão seria uma abordagem pouco benéfica, comparativamente ao uso da complexidade.

Na Tabela 3.1, são apresentados valores médios de complexidade. Por cada célula da tabela, foram gerados pseudo-aleatoriamente 500 processos (a descrição deste método pseudo-aleatório é aprofundada no capítulo 4), com o número de participantes e de elementos (incluindo os mutáveis e fixos) indicados, e calculada a média das complexidades dos processos. Percebemos pelos valores calculados que à medida que o número de elementos e participantes aumenta, mais desproporcional será o número total de soluções.

O tempo de gerar o conjunto ótimo de um processo com apenas 15 elementos e três participantes (complexidade de algumas dezenas de milhares) será muito menor do que para um processo com 30 elementos e três participantes, que deverá ter, na maioria das vezes, várias centenas ou milhares de milhões de alternativas ao modelo original.

| | 2 Participantes | 3 Participantes | 4 Participantes |
|---|---|---|---|
| 10 Elementos | 62 | 1153 | 10936 |
| 15 Elementos | 470 | 34022 | 931423 |
| 20 Elementos | 3238 | 1238785 | 69104915 |
| 25 Elementos | 27985 | 20862232 | 8188015839 |
| 30 Elementos | 178025 | 1288663874 | 458237235757 |
| 40 Elementos | 10935787 | 796506069342 | 3119283394027388 |

Tabela 3.1 Média de complexidades das instâncias criadas pseudo-aleatoriamente

## 3.3 Algoritmo de busca exaustiva

Para se proceder à análise dos resultados das meta-heurísticas, implementou-se um algoritmo de busca exaustiva, ou busca por força bruta, que encontra todas as soluções ótimas de um processo. A Figura 3.1 mostra o seu pseudo-código. O algoritmo começa por uma gerar uma lista *C* com todas as combinações do processo (cada solução será gerada a partir de uma combinação), e adiciona a solução do modelo original do processo, *curr*, à lista *L*. Em cada iteração, uma nova solução *new* é gerada. Se *new* for dominada por *curr* (ou igual a *curr*), *new* é descartada, e avança-se para a iteração seguinte. Caso contrário, a lista *L* é corrida e todas as soluções dominadas por *new* são removidas. Se nenhuma solução restante em *L* dominar *new*, esta última é adicionada a *L*, e *curr* passa a ser igual a *new*. Após a execução terminar, as soluções contidas em *L* serão as soluções ótimas do processo, contidas na fronteira de Pareto.

```
L ← {}
C ← generate_combinations()
curr ← initial_solution()//added to L
foreach c ∈ C
        new ← generateSolution(c)
        if !dominates(curr, new) or equals(curr, new)
                all solutions in L dominated by new are removed
                if no solution in L dominates new
                        new is added to L
                        curr ← new
                end
        end
end
```

Figura 3.1 Pseudo-código do algoritmo de busca exaustiva

**Processo Ilustrativo**

O processo de irrigação é composto por dois participantes (*pool* do sistema central e *pool* dos aparelhos *IoT*) e sete elementos mutáveis. Podemos, por isso, deduzir que cada um desses sete elementos poderá ser executado, no máximo, em duas *pools* diferentes. Posto isto, a complexidade do processo será de $2^7 = 128$ soluções.

O algoritmo de busca exaustiva foi executado sob o processo e terminou com um conjunto final de três soluções ótimas, com os seguintes valores arredondados de fiabilidade e custo da comunicação, respetivamente.

- S1 (0.48375, 0.82463)
- S2 (0.43712, 0.85075)
- S3 (0.39499, 0.87687)

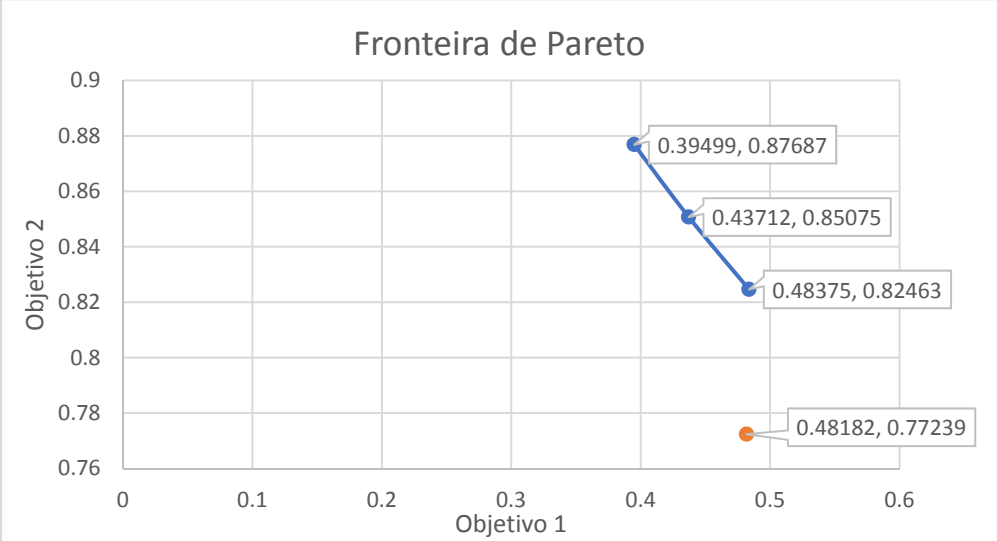


Figura 3.2 Fronteira de Pareto do processo de irrigação

Notamos que nenhuma delas corresponde à solução que descreve o modelo inicial do processo, avaliada em 3.1, com o par de valores (0.48182, 0.77239) para os objetivos. Embora a solução não domine nem seja dominada por S2 e S3, o mesmo já não acontece com S1, que tem valores de fiabilidade e contraparte do custo superiores à desta solução. Como tal, esta situa-se, não na linha da fronteira de Pareto, mas abaixo dessa, como se pode observar pela Figura 3.2.

De realçar que, enquanto o processo inicial apresenta um total de sete comunicações, verificou-se que as soluções S1, S2 e S3 contêm (e com sentido) apenas cinco, quatro e três trocas de mensagens entre *pools*, respetivamente. Podemos, assim, concluir que, de

entre as soluções ótimas do processo, a solução S1 é a mais fiável, enquanto a solução S3 é a menos dispendiosa.

No caso deste exemplo, a execução do algoritmo foi rápida, pois o tempo de gerar as 128 soluções, somado ao tempo de ir atualizando o conjunto não-dominado é inferior a um segundo. Tratando-se de um processo de complexidade muito reduzida, e que requer pouco esforço computacional, não há necessidade de se considerar o uso de algoritmos meta-heurísticos para este caso.

Por fim, podemos ainda determinar o valor da área do poliedro definido pelo conjunto de soluções ótimas. Esta área é, como explicado em 2.5.4, a métrica S ou *hypervolume* de Zitzler [33]. Num problema de dois objetivos, como é o caso, o valor calculado é uma área, já que a fronteira de Pareto insere-se num espaço de duas dimensões. De três objetivos em diante, o valor a determinar passa a ser um volume. Para a determinação da área resultante do conjunto ótimo, a origem (0,0) foi usada como o ponto de referência, e o valor determinado foi de aproximadamente 0.42065. Esta é, por isso, a maior área que um algoritmo pode obter relativamente a este processo, visto que representa o grupo das soluções mais eficientes do mesmo. Como tal, quanto mais próxima a área de um conjunto encontrado por uma meta-heurística estiver da área do conjunto ótimo, mais confiável essa meta-heurística será na substituição de algoritmos exatos.

A Figura 3.3 mostra, para este exemplo, o resultado da execução do algoritmo. O ficheiro apresenta os valores das soluções ótimas do processo, número de soluções, tempo de execução do algoritmo (em milissegundos), e o valor da área do conjunto. Por as soluções não se dominarem entre elas, os valores de fiabilidade e custo encontram-se sempre por ordem decrescente e crescente, respetivamente.

```
1  Time
2  123
3
4  Optimal solutions
5  3
6
7  Reliability, Cost Inverse
8  0.483748665169,0.8246268656716418
9  0.437119656308,0.8507462686567164
10 0.394985263398,0.8768656716417911
11
12 Area
13 0.4206462292548172
```

Figura 3.3 Resultado com as soluções ótimas do processo de irrigação automática

## 3.4 Algoritmos meta-heurísticos desenvolvidos

A seguir, são descritas as implementações dos dois algoritmo meta-heurísticos desenvolvidos para a otimização bi-objetivo do problema em estudo.

Por várias vezes, vão ser mencionados os termos *solução atual* e *mutação*. Solução atual é a solução onde a procura se situa atualmente no espaço de soluções. Mutação é uma nova solução gerada na vizinhança da solução atual. Esta consiste na alteração de participante de um só elemento mutável da solução atual. É, ainda, usado o termo *mutação total* – esta refere-se, também, a uma nova solução gerada a partir da atual, com a diferença que todos os elementos mutáveis do processo, traduzido pela solução atual, são alterados quanto ao seu participante. Ou seja, quanto maior o espaço de soluções, mais provável será de ocorrer uma mudança radical da posição da pesquisa no espaço de soluções.

### 3.4.1 *Local Search*

De seguida, é descrita a implementação do algoritmo baseado em pesquisa local, como se ilustra na Figura 3.4.

No início da execução começa-se por gerar uma mutação total. Isto significa que todos os elementos mutáveis do modelo original do processo são alterados quanto ao participante que os executa. Em cada iteração, é gerada uma mutação. A ideia genérica de algoritmos como o *Local Search* sugere que a solução atual apenas deve ser alterada se for dominada pela mutação. Na implementação deste algoritmo o critério foi alargado, bastando à mutação não ser dominada pela atual, para que ocorra a transição de estado. O objetivo dessa mudança é potenciar a busca, de modo a diminuir as possibilidades de ficar presa num extremo local. Ao longo da execução, é mantida uma lista com as soluções mais eficientes até ao momento. Sempre que a solução atual é alterada, esta é comparada às soluções da lista. Se dominar alguma solução da lista, esta última é removida. De seguida, são contadas as soluções da lista que dominam a atual. Se nenhuma delas a dominar, a solução atual é adicionada à lista. No fim da execução, as soluções mantidas na lista constituirão a fronteira de Pareto.

```
initialize max_iterations, max_stable_iterations
L ← {}
curr ← total_mutation() //added to L
it ← 0
stable_it ← 0
while it < max_iterations and stable_it < max_stable_iterations
        it ← it + 1
        new ← mutate(curr)
        if !dominates(curr, new) then
                curr ← new
                all solutions in L dominated by curr are removed
                if no solution in L dominates curr then
                        curr is added to L
                        stable_it ← 0
                elseif
                        stable_it ← stable_it + 1
        elseif
                stable_it ← stable_it + 1
end
```

Figura 3.4 Pseudo-código do algoritmo *Local Search* para o problema de otimização bi-objetivo da descentralização da computação

O algoritmo apresenta duas condições de paragem: (1) executar o número máximo de iterações ou (2) executar um determinado número de iterações consecutivas sem melhoria. A condição *it* < *maxIt* segue o critério baseado na exaustão (1), e força o algoritmo a terminar a execução ao fim de *maxIt* iterações. O número escolhido depende do tempo que estamos dispostos a investir na execução, que pode variar, e muito, consoante o número de objetivos ou tamanho do espaço de soluções. A condição *stable_it* < *max_stable_it* segue o critério baseado na melhoria (2), e indica que ao fim de *maxItStable* iterações estáveis consecutivas (tipicamente um valor bem abaixo de *maxIt*), a procura é concluída. Uma iteração é estável em duas situações:

a) A solução atual não é alterada, porque domina a mutação gerada.
b) A solução atual é alterada, mas a nova solução respetiva não é adicionada ao conjunto não dominado.

Em nenhum dos dois casos anteriores a fronteira de Pareto foi modificada, daí o número de iterações estáveis consecutivas ser incrementado. Ao se atingir o limite *maxItStable*, o algoritmo transmite a ideia de que a descoberta de novas soluções *estabilizou*, terminando assim a execução antes de atingir o valor *maxIt*. Se a procura encontrasse uma nova solução não dominada, o valor *stable_it* recomeçaria no zero, e a procura prosseguiria durante, pelo menos, mais *maxItStable* iterações (a não ser que o valor *maxIt* estivesse mais próximo de se atingir). Se a situação a) se repetir por várias

iterações seguidas, é provável que o algoritmo tenha convergido para um extremo local, do qual não consegue escapar. Nesse caso, o algoritmo pode parar porque, garantidamente, não encontrará uma solução vizinha melhor (ou no mínimo, que não seja pior). Um grande número de ocorrências de b) sugere a dificuldade da procura em descobrir novas soluções não dominadas.

### 3.4.2 *Simulated Annealing*

A implementação multi-objetivo do *Simulated Annealing* aqui apresentada é baseada no algoritmo SMOSA [28], explicado previamente. Tal como no *Local Search*, o algoritmo começa por gerar uma mutação total sobre a solução correspondente ao processo original.

Durante as primeiras $N_{T1}$ iterações, todas as transições de estado são aceites, ou seja, a solução atual é sempre alterada. Em cada iteração, o valor de fiabilidade (Objetivo 1) da nova solução é adicionado à lista de fiabilidades, e o mesmo se sucede para o valor do complemento do custo (Objetivo 2). Alcançadas as $N_{T1}$ iterações, são calculados os desvios padrão, $\sigma_{fiabilidade}$ e $\sigma_{custo}$, das respetivas listas. As temperaturas iniciais são então inicializadas: $T_{fiabilidade} = \sigma_{fiabilidade}$ e $T_{custo} = \sigma_{custo.}$ Daí em diante, até que a execução termine, a probabilidade de ocorrer a mudança de estado é o produto das probabilidades de aceitação dos dois objetivos, tal como explicado na secção 2.5.2. As temperaturas são reduzidas a cada $N_{T2}$ iterações ou $N_A = 0.4 \times N_{T2}$ transições de estado aceites, como sugerido pelos autores. Ambos os valores se mantêm constantes, cada vez que são alcançados. Para cada objetivo, a nova temperatura *T'* é reduzida pela seguinte fórmula:

$$T' = T \times \alpha \tag{9}$$

onde $T$ é a temperatura atual e $\alpha$ é determinado por:

$$\alpha = max\left(0.5, exp\left[-\frac{0.7 \times T}{\sigma}\right]\right) \tag{10}$$

sendo $\sigma$ o desvio padrão do conjunto de valores do objetivo de todas as soluções aceites até ao momento, à respetiva temperatura $T$.

A estratégia de retorno à base é também empregue. Tal significa que, de um dado período em período de iterações (e cujo número vai diminuindo, sempre que é atingido),

o algoritmo recomeça a procura, não na solução atual (ou numa mutação dessa), mas numa solução do conjunto não dominado. A solução escolhida é sempre uma das três hipóteses: solução num dos extremos (a de maior fiabilidade), solução no outro extremo (a de menor custo), ou a solução mais isolada em relação a todas as outras soluções do conjunto. Na hipótese da solução mais isolada, os extremos não são candidatos à escolha.

Outra opção possível, além do retorno à base, é gerar uma mutação total. Ambas as abordagens têm a sua vantagem. A estratégia de retorno à base faz com que a procura recomece na vizinhança duma solução que é, até ao momento, não dominada. A segunda opção força a procura a recomeçar numa zona do espaço de soluções que pode ter sido pouco, ou nunca, explorada durante a execução.

Há por isso, três casos considerados, e usados sempre pela mesma ordem quando o período é atingido. Da primeira vez que esse é alcançado, é escolhida a solução mais isolada do conjunto não dominado, e a contagem de iterações no período recomeça. Na segunda vez, a procura reinicia-se num dos extremos do conjunto. Na terceira vez, é escolhida uma mutação total. De referir ainda que, no caso de escolha da solução mais isolada, se o conjunto não dominado contiver menos de três soluções, ao invés de se escolher um extremo desse conjunto, gera-se novamente uma mutação total. Ao longo da execução, o número de iterações para ocorrer uma dessas opções é diminuído sempre que esse é alcançado, com um taxa igual a 0.9. Por exemplo, se numa dada altura da execução o período for igual a 400, quando a procura alcançar esse número de iterações, o período passa a ser de 360. Na Figura 3.5 é mostrado o pseudo-código do algoritmo.

```
initialize max_iterations, NT1
L ← {}
curr ← total_mutation() //added to L
it ← 0
accept first NT1 mutations //curr is always changed
keep two records with the observed objective function values
temperature ← initial_temperature() //use records
while (it < max_iterations)
        it ← it + 1
        new ← mutate(curr)
        if archived(new) //if new is non dominated
                curr ← new
        elseif acceptanceProbability
                curr ← new
        update records
        periodically returnToBase() or total_mutation()
        periodically reduce temperature //use records
end
```

Figura 3.5 Pseudo-código do algoritmo *Simulated Annealing* para o problema de otimização bi-objetivo da descentralização da computação

## 3.5 Execução de tarefas

A resolução do problema de otimização para um determinado caso de uso deve, tipicamente, consistir da seguinte ordem proposta de tarefas:

1. Elaboração de um processo BPMN em representação gráfica, sob forma de diagrama. O código fonte é apresentado na linguagem XML.
2. Um *software* interpreta o código XML do processo BPMN e são criadas estruturas de dados que o representam.
3. Um algoritmo é implementado com vista a encontrar as soluções descentralizadas que satisfazem as restrições e objetivos do problema de otimização.
4. As estruturadas de dados que representam as soluções encontradas são convertidas em linguagem XML, de modo a serem representadas sob forma de diagrama, tal como no modelo original do processo elaborado no primeiro passo.

Neste trabalho, para os testes aos algoritmos implementados serem realizados, foi usado um método disponibilizado que gera processos BPMN automaticamente, que será referido no capítulo 4. Ao invés de estes serem criados na linguagem XML, é gerado um

ficheiro .CSV, que é posteriormente traduzido nas estruturas de dados referidas no passo 2. Este método possibilita que um maior número de processos sejam utilizados nos testes, dado que a criação por meio de elaboração gráfica pode ser demorada. Devido ao tempo limitado, não foi possível converter as estruturas de dados, com as soluções encontradas, para código XML, de modo a serem graficamente representadas, como explicado no passo 4.

# Capítulo 4 Testes computacionais

Neste capítulo, são descritos os testes efetuados usando um conjunto de instâncias geradas de forma pseudo-aleatória, por parte dos algoritmos desenvolvidos, de forma a avaliar a qualidade dos seus resultados. Os testes e medições de tempo foram efetuados numa máquina com processador Intel Core i5-3337U, 1,8 GHz e 6GB de memória RAM, no sistema operativo Windows 10.

## 4.1 Criação de instâncias

Para analisar a qualidade dos algoritmos, foi necessário gerar um conjunto de processos BPMN para a realização de vários testes. Existem dois caminhos para a criação de processos: elaboração manual e método automático.

A via manual realiza-se pela elaboração de um diagrama BPMN, que requer a criação de *pools* (e respetivas *lanes*, se necessário), elementos que representam os objetos de fluxo e objetos de ligação entre os elementos, que determinam a ordem de execução das atividades. A elaboração do diagrama pode ser feita através de uma ferramenta de modelação gráfica de processos ou por edição direta do código fonte do ficheiro, na linguagem XML. O valor de fiabilidade de cada um dos elementos deve também ser introduzido, bem como a probabilidade de serem escolhidos, se pertencerem a uma *gateway* exclusiva.

O método automático permite gerar instâncias pseudo-aleatórias de processos em BPMN considerando alguns parâmetros que definem a estrutura e dimensão do processo. Este programa permite criar um processo .CSV (ficheiro separado por vírgulas) a partir dos dados de *input* lidos de um ficheiro de configuração. O programa de geração automática começa por receber o ficheiro de *input* com a configuração desejada para o processo, cria as estruturas de dados necessárias e converte a informação para o ficheiro de *output* em .CSV. O processo contrário pode também ser realizado: o programa lê e traduz um ficheiro .CSV que representa um processo em BPMN para estruturas de dados.

De seguida, é mostrado um exemplo do ficheiro de configuração para um processo com dois participantes, na Figura 4.1. O ficheiro de *input* mantém o formato abaixo para qualquer processo a ser criado:

- Linha 1: número de *pools* (participantes) do processo. Estas são identificadas com valores inteiros. Por exemplo, se o processo tiver quatro *pools*, elas serão identificadas por 1, 2, 3 e 4.
- Linha 2: número de elementos (objetos de fluxo) do processo. Este valor não inclui as tarefas de envio e receção de mensagens (representadas pela notação ReceiveTask e SendTask, como veremos a seguir).
- Linha 3: precisão de apresentação dos valores gerados.
- Linha 4: probabilidade de o elemento gerado ser uma *gateway*.
- Linha 5: probabilidade de um elemento convergir, se este for uma *Complex Gateway*.
- Linhas 6 até 7: intervalo onde devem ser gerados os valores de fiabilidade para os elementos em cada *pool*. Cada linha corresponde a uma *pool*.
- Linhas 8 até 9: fiabilidade da transmissão de uma mensagem da *pool* 1 para a *pool* 2, e *pool* 2 para a *pool* 1, respetivamente. Para um processo com n *pools* seriam preenchidas n×(n-1) linhas com os respetivos valores de fiabilidade de transmissão entre cada par de linhas.
- Linhas 10 até 11: custo da transmissão de uma mensagem da *pool* 1 para a *pool* 2, e *pool* 2 para a *pool* 1, respetivamente. Para um processo com n *pools* seriam preenchidas n×(n-1) linhas com os respetivos valores de custo de transmissão entre cada par de linhas.

```
1 2
2 7
3 3
4 0.7
5 0.8
6 1-0.998-0.999
7 2-0.997-0.998
8 1-2=0.99
9 2-1=0.98
10 1-2=7
11 2-1=13
```

Figura 4.1 Ficheiro de *input* para a geração de uma instância pseudo-aleatória de BPMN com duas *pools*

A Figura 4.2 apresenta um exemplo de ficheiro .CSV que representa um processo gerado pseudo-aleatoriamente, com os dados do ficheiro de configuração da Figura 4.1.

Todas as linhas apresentam o mesmo número de colunas, com igual significado, e cada uma corresponde à informação de um elemento, com as seguintes características:

- **Nome** do elemento.
- **Pool** atual do elemento.
- **F** fiabilidade do elemento.
- **Prob** indica o quão provável é um elemento contido numa *gateway* do tipo Exclusiva ser a alternativa escolhida. O processo de exemplo tem dois elementos que definem o início e fim de uma *gateway* exclusiva (Exclusive Gateway 1 e Exclusive Gateway 1 END). Observamos que a *gateway* tem duas alternativas possíveis de escolha, como se vê pela coluna dos elementos sucessores de Exclusive Gateway 1: Task 3 e Task 4. Verifica-se, corretamente, que a soma das probabilidades desses dois elementos é igual a um. Como os restantes elementos não pertencem à *gateway*, têm valor igual a zero.
- **C** é utilizado para *gateways* de tipo complexas e indica o número mínimo de elementos K que devem ser escolhidos de entre N elementos possíveis contidos nessa *gateway*. Por não houver nenhuma *gateway* desse tipo neste exemplo, todos os valores são zero.
- **Sucessor** indica os sucessores do elemento.
- **Pools** indica os participantes pelos quais o elemento pode ser executado, separados por '/'.
- **F-Pool** corresponde à fiabilidade do elemento conforme a *pool* onde for executado.

Por fim, podemos realçar alguns aspetos identificáveis:

- Existem duas mensagens trocadas entre os dois participantes do processo.
- Task 3, Task 4, e Task 5 são os elementos mutáveis do processo.
- Os elementos Start Event 1, End Event 1 e End Event 2 são omitidos do ficheiro .CSV, enquanto que Start Message Event 1 mantém-se no ficheiro .CSV porque indica a receção de uma mensagem, e é designado de Receive Task 1.

| Nome | Pool | F | Prob | C | Sucessor | Pools | F-Pool |
|---|---|---|---|---|---|---|---|
| Task 1 | 1 | 0.999 | 0 | 0 | Send Task 1 | 1 | 1-0.999 |
| Send Task 1 | 1 | 0.99 | 0 | 0 | Receive Task 1/Receive Task 2 | 1 | 1-0.99 |
| Receive Task 1 | 2 | 1 | 0 | 0 | Task 2 | 2 | 2-1.0 |
| Task 2 | 2 | 0.997 | 0 | 0 | Send Task 2 | 2 | 2-0.997 |
| Send Task 2 | 2 | 0.98 | 0 | 0 | Receive Task 2 | 2 | 2-0.98 |
| Receive Task 2 | 1 | 1 | 0 | 0 | Exclusive Gateway 1 | 1 | 1-1.0 |
| Exclusive Gateway 1 | 1 | 0.998 | 0 | 0 | Task 3/Task 4 | 1 | 1-0.998 |
| Task 3 | 1 | 0.998 | 0.204673919 | 0 | Exclusive Gateway 1 END | 1/2 | 1-0.998/2-0.997 |
| Task 4 | 1 | 0.999 | 0.795326081 | 0 | Exclusive Gateway 1 END | 1/2 | 1-0.999/2-0.998 |
| Exclusive Gateway 1 END | 1 | 0.998 | 0 | 0 | Task 5 | 1 | 1-0.998 |
| Task 5 | 1 | 0.999 | 0 | 0 | | 1/2 | 1-0.999/2-0.998 |

Figura 4.2 Ficheiro CSV do processo de exemplo

A Figura 4.3 apresenta o mesmo processo, em formato gráfico, tendo-se recorrido à ferramenta BPMN2 *Modeler* para a sua elaboração no ambiente de desenvolvimento integrado Eclipse [4].

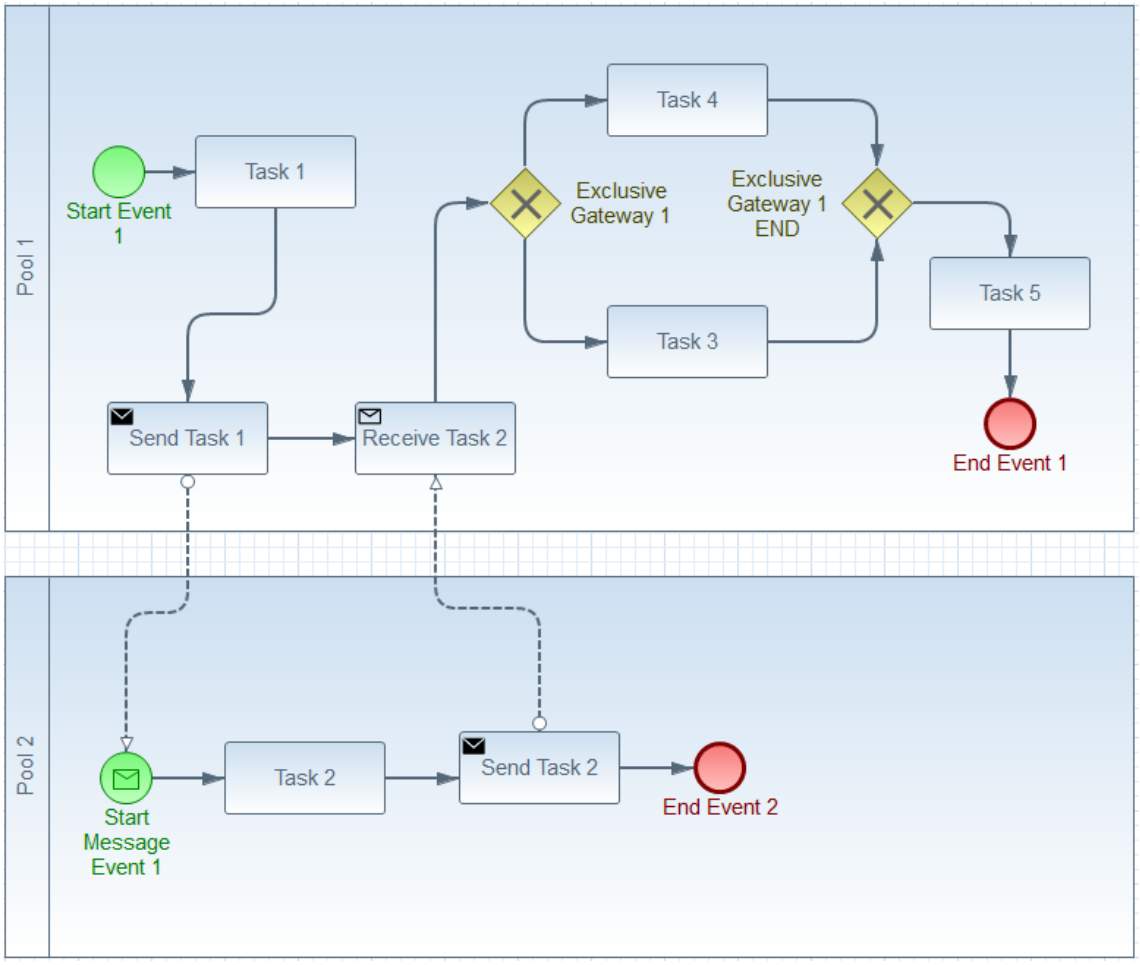


Figura 4.3 Representação gráfica em BPMN do processo de exemplo

## 4.2 Instâncias utilizadas e resultados da busca exaustiva

Recorrendo ao método de geração pseudo-aleatória, foram gerados 30 processos com a finalidade de se avaliar a qualidade dos algoritmos de otimização. Os 30 processos apresentam complexidades variadas, e algumas delas consideravelmente grandes (vão desde aproximadamente 500 mil às 60 milhões de soluções), de forma a que o espaço de soluções de cada um deles fosse grande o suficiente para tornar o trabalho dos algoritmos de otimização o mais difícil possível. Após a geração dos processos, executou-se o algoritmo de busca exaustiva, registando-se o conjunto de soluções ótimas encontradas em cada um. Determinar o conjunto ótimo de cada processo exigiu um esforço computacional de vários minutos ou até horas, conforme a respetiva complexidade de cada um. O tempo de gerar todas as soluções, bem como atualizar o conjunto ótimo ao longo da execução justificam esse esforço.

O facto de o gerador automático escolher aleatoriamente os elementos mutáveis fez com que, por várias vezes, os valores de complexidade dos processos criados fossem extremamente elevados. Ou seja, pode certamente acontecer que, para dois processos com igual número de elementos, a parcela correspondente aos mutáveis seja diferente. Uma pequena diferença no número (gerado) de elementos aptos a mudar de *pool* pode resultar em valores de complexidade muito distintos. Um processo com complexidade demasiado grande para o tempo de análise disponível seria posteriormente descartado para esta análise, sendo preferível um processo que tivesse uma complexidade praticável.

De entre os 30 processos, quatro deles têm duas *pools*, dezoito têm três *pools* e os oito restantes têm quatro *pools*. Na criação dos processos em formato .CSV, foram utilizados três ficheiros de configuração distintos como *input*, um por cada número diferente de *pools* do processo, como ilustrado na Figura 4.4. Para cada ficheiro de configuração, apenas o número de elementos foi alterado. Para processos de duas *pools*, o número de elementos variou entre os valores 38, 40 e 50. Em processos de três *pools* utilizou-se os valores 25, 26 e 27. Por fim, processos de quatro *pools* têm 22 ou 23 elementos. É natural que à medida que o número de *pools* aumentava, o número introduzido de elementos fosse diminuindo, para evitar possíveis complexidades demasiado elevadas, pelo motivo já referido.

Os ficheiros de configuração utilizados são exibidos nas figuras abaixo. Apenas a linha 2, correspondente ao número de elementos, teve uma variação de valores. Os

restantes parâmetros de cada ficheiro foram utilizados sempre com os mesmos valores para os processos correspondentes: número de *pools* (linha 1), precisão dos valores gerados (linha 3), probabilidade de o elemento gerado ser uma *gateway* (linha 4), probabilidade de um elemento convergir se for uma *gateway* do tipo complexa (linha 5), intervalo de fiabilidades por cada *pool* (cada linha inclui dois valores *Double*), fiabilidade da transmissão entre cada duas *pools* (um valor *Double* por transmissão) e custo da transmissão entre cada duas *pools* (um valor inteiro por transmissão).

```
2
<Elementos>
3
0.7
0.8
1-0.998-0.999
2-0.997-0.998
1-2=0.99
2-1=0.98
1-2=7
2-1=13
```

```
3
<Elementos>
3
0.7
0.8
1-0.98-0.99
2-0.98-0.99
3-0.998-0.999
1-2=0.99
2-1=0.99
1-3=0.999
3-1=0.999
2-3=0.995
3-2=0.995
1-2=2
2-1=2
1-3=16
3-1=16
2-3=8
3-2=8
```

```
4
<Elementos>
3
0.7
0.8
1-0.998-0.999
2-0.997-0.998
3-0.997-0.999
4-0.996-0.997
1-2=0.99
1-3=0.99
1-4=0.99
2-1=0.98
2-3=0.99
2-4=0.99
3-1=0.98
3-2=0.99
3-4=0.99
4-1=0.98
4-2=0.99
4-3=0.99
1-2=5
1-3=7
1-4=9
2-1=10
2-3=8
2-4=7
3-1=7
3-2=6
3-4=12
4-1=10
4-2=9
4-3=6
```

Figura 4.4 Ficheiros de configuração para a geração de processos

A Tabela 4.1 apresenta as características de cada instância. São ainda exibidos os resultados da execução do algoritmo de busca exaustiva para cada uma delas. Os resultados incluem o tempo de execução do algoritmo, o número de soluções do conjunto ótimo, e o valor da área calculada para esse conjunto.

- **ID**: valor único de identificação do processo/instância.
- **NrP**: número total de *pools* (participantes).
- **NrE**: número de elementos (objetos de fluxo). Estes podem consistir em tarefas, eventos ou *gateways*. Apenas as tarefas relacionadas com a realização de atividades são incluídas nesta contagem, enquanto que as tarefas de envio e receção de mensagens (representadas pela notação SendTask e ReceiveTask, respetivamente) são excluídas.
- **NrEM**: número de elementos que podem ser executados por mais do que um participante.
- **NrC**: número total de comunicações ocorridas entre diferentes participantes.
- **Comp**: complexidade do processo. Este valor corresponde à dimensão do espaço de soluções.
- **T(min)**: tempo, em minutos, levado a cabo pelo algoritmo de busca exaustiva na geração do conjunto de soluções ótimas do processo.
- **NrO**: número de soluções ótimas.
- **A**: valor da área definida pelo conjunto de soluções ótimas, calculada com base na métrica *Hypervolume* [33].

| ID | NrP | NrE | NrEM | NrC | Comp | T(min) | NrO | A |
|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 25 | 18 | 24 | 22674816 | 85 | 3 | 0.88972 |
| 2 | 3 | 25 | 16 | 17 | 2519424 | 6 | 5 | 0.81554 |
| 3 | 3 | 25 | 19 | 22 | 8957952 | 26 | 6 | 0.69084 |
| 4 | 4 | 22 | 16 | 20 | 11943936 | 32 | 3 | 0.73262 |
| 5 | 4 | 22 | 15 | 22 | 13436928 | 40 | 3 | 0.82027 |
| 6 | 4 | 22 | 14 | 19 | 15925248 | 46 | 1 | 0.82201 |
| 7 | 4 | 22 | 14 | 15 | 4718592 | 11 | 1 | 0.72252 |
| 8 | 4 | 23 | 16 | 23 | 53747712 | 166 | 2 | 0.78904 |
| 9 | 4 | 23 | 16 | 16 | 11943936 | 33 | 2 | 0.74487 |
| 10 | 4 | 23 | 17 | 19 | 56623104 | 160 | 1 | 0.76982 |
| 11 | 3 | 25 | 19 | 23 | 45349632 | 140 | 8 | 0.80999 |
| 12 | 3 | 25 | 18 | 20 | 15116544 | 50 | 5 | 0.7839 |
| 13 | 3 | 26 | 19 | 19 | 30233088 | 95 | 6 | 0.82175 |
| 14 | 3 | 26 | 18 | 19 | 34012224 | 98 | 7 | 0.75165 |
| 15 | 3 | 26 | 19 | 19 | 20155392 | 57 | 6 | 0.8123 |
| 16 | 3 | 26 | 19 | 20 | 30233088 | 88 | 15 | 0.86858 |
| 17 | 3 | 26 | 20 | 22 | 11943936 | 40 | 9 | 0.89432 |
| 18 | 3 | 26 | 16 | 16 | 8503056 | 27 | 3 | 0.84538 |
| 19 | 3 | 27 | 20 | 23 | 60466176 | 206 | 20 | 0.79197 |
| 20 | 3 | 27 | 20 | 21 | 60466176 | 222 | 13 | 0.90507 |
| 21 | 3 | 27 | 19 | 23 | 30233088 | 98 | 7 | 0.84111 |
| 22 | 3 | 27 | 19 | 19 | 20155392 | 77 | 4 | 0.82795 |
| 23 | 3 | 27 | 19 | 19 | 8957952 | 26 | 8 | 0.75296 |
| 24 | 3 | 27 | 18 | 20 | 10077696 | 35 | 6 | 0.81481 |
| 25 | 3 | 27 | 17 | 20 | 2239488 | 7 | 7 | 0.90198 |
| 26 | 4 | 23 | 17 | 24 | 31850496 | 96 | 1 | 0.89401 |
| 27 | 2 | 50 | 20 | 27 | 1048576 | 4 | 4 | 0.74469 |
| 28 | 2 | 38 | 19 | 20 | 524288 | 2 | 5 | 0.88641 |
| 29 | 2 | 40 | 20 | 16 | 1048576 | 3 | 1 | 0.85228 |
| 30 | 2 | 40 | 21 | 23 | 2097152 | 7 | 1 | 0.84137 |

Tabela 4.1 Instâncias pseudo-aleatórias e resultados da busca exaustiva

Ao se ordenar os processos por ordem crescente do grau de complexidade (conforme Anexo A), observamos que, na maioria dos casos, à medida que a complexidade de um processo aumenta, o tempo de computação da busca exaustiva aumenta também. O crescimento não é puramente linear, pois notamos que há processos com menor complexidade que outros, embora com um tempo de execução superior. O

principal motivo será pelos processos apresentarem diferentes configurações dos seus elementos, fazendo com que o tempo de gerar uma solução (cálculo da fiabilidade e contraparte do custo) também varie. Admite-se, claro, que em processos de complexidade muito reduzida, a diferença entre os tempos de gerar todas as soluções seja de apenas alguns milissegundos. Contudo, para os cenários onde o tamanho do espaço de soluções se situa na casa dos milhões, é natural que a diferença se faça notar numa escala de minutos.

## 4.3 Resultados

Para cada processo, foram executados 30 *runs* de cada meta-heurística, e calculadas as respetivas médias. A escolha de se executarem 30 *runs* baseia-se na possibilidade de os algoritmos poderem gerar resultados bastante díspares entre diferentes execuções do mesmo processo, por não serem determinísticos. Mesmo que existam processos onde a maioria dos *runs* possa ter igual número de soluções ótimas encontradas, seria mais difícil tirar conclusões fiáveis, com um número reduzido de testes por processo. Assim, independentemente de se obter, ou não, uma maior variedade de resultados, o número escolhido de *runs* contribui para uma avaliação mais precisa do desempenho dos algoritmos.

Convém realçar que quando é usado o termo *conjunto aproximado* (ou *soluções aproximadas*), este refere-se sempre às soluções obtidas pelas meta-heurísticas, enquanto que *conjunto ótimo* (ou *soluções ótimas*) diz respeito ao conjunto de soluções determinado pela busca exaustiva.

Ambos os algoritmos foram executados com o mesmo número total de iterações, 4000. Para o LS, foi ainda necessário introduzir como parâmetro o número de iterações estáveis, que definimos como 400. Embora este valor seja bastante inferior ao primeiro, tal apenas serve para evitar que o algoritmo continue a procura após 400 iterações consecutivas sem atualizar o conjunto aproximado. No SA, foi apenas necessário introduzir o número máximo de iterações, dado que os valores de temperatura inicial e taxa de arrefecimento são calculados de acordo com os dados da procura ao longo da execução. Por isso, e como já referido no capítulo 3, ambos os algoritmos apresentam um critério de paragem baseado na exaustão, e o *Local Search* tem adicionalmente um critério baseado na melhoria. Visto que os conjuntos ótimos de todas as instâncias geradas já eram

conhecidos à partida, poder-se-ia ter ponderado a hipótese de os algoritmos implementados seguirem o critério baseado na referência. Assim, a execução podia até terminar antecipadamente, ao saber que se tinha encontrado o conjunto ótimo na totalidade, ou que se encontrava próximo desse (conforme o pretendido pelo *designer*). No entanto, as meta-heurísticas implementadas destinam-se a ser usadas em futuras instâncias para as quais se desconhece as soluções ótimas à partida, devido aos valores de complexidade poderem ser demasiado grandes para um algoritmo de busca exaustiva terminar em tempo aceitável.

Foram elaboradas duas tabelas que apresentam os resultados dos algoritmos executados para cada um dos processos (encontram-se também em anexo, por ordem crescente de complexidade dos processos). A Tabela 4.2 tem um igual número de colunas, com o mesmo significado, tanto para o SA como para o LS e cada uma representa valores médios dos 30 *runs*.

As colunas da tabela são as seguintes:

- **NrO**: número de soluções ótimas do processo (da Tabela 4.1)
- **T(ms)** é o tempo de execução do algoritmo, em milissegundos
- **NrS** tamanho do conjunto aproximado
- **Área** do conjunto aproximado
- **Dif** diferença entre a área do conjunto aproximado e o conjunto ótimo
- **NrOE** número de soluções do conjunto aproximado contidas no conjunto ótimo
- **% Ótimas** percentagem de soluções do conjunto aproximado contidas no conjunto ótimo (cobertura)

Por uma questão de legibilidade, os valores das áreas dos conjuntos aproximado obtidos pelos algoritmos e das diferenças para as áreas ótimas foram arredondados para cinco casas decimais, ao serem copiados para as tabelas. Além disso, para manter a precisão, as diferenças entre as áreas dos conjuntos ótimos e aproximados obtidos pelos algoritmos foram calculadas com os valores reais, e só posteriormente se procedeu ao arredondamento.

| ID | NrO | SA | | | | | | LS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T(ms) | NrS | Área | Dif | NrOE | % Ótimas | T(ms) | NrS | Área | Dif | NrOE | % Ótimas |
| P1 | 3 | 401 | 2.70 | 0.88666 | 0.00306 | 2.67 | 0.89 | 209 | 2 | 0.88035 | 0.00937 | 2 | 0.67 |
| P2 | 5 | 272 | 5 | 0.81554 | 0 | 5 | 1 | 191 | 5 | 0.81554 | 0 | 5 | 1 |
| P3 | 6 | 293 | 6 | 0.69084 | 0 | 6 | 1 | 143 | 5.90 | 0.68482 | 0.00602 | 3.27 | 0.54 |
| P4 | 3 | 275 | 2.8 | 0.73105 | 0.00157 | 2.8 | 0.93 | 244 | 1.87 | 0.72062 | 0.012 | 1.33 | 0.44 |
| P5 | 3 | 296 | 3 | 0.82027 | 0 | 3 | 1 | 266 | 3 | 0.82027 | 0 | 3 | 1 |
| P6 | 1 | 330 | 1 | 0.82201 | 0 | 1 | 1 | 274 | 1.03 | 0.81670 | 0.00531 | 0.67 | 0.67 |
| P7 | 1 | 268 | 1 | 0.72252 | 0 | 1 | 1 | 295 | 1 | 0.72252 | 0 | 1 | 1 |
| P8 | 2 | 332 | 2 | 0.78904 | 0 | 2 | 1 | 338 | 2 | 0.78904 | 0 | 2 | 1 |
| P9 | 2 | 302 | 2.27 | 0.74375 | 0.00111 | 1.50 | 0.75 | 365 | 2.13 | 0.73440 | 0.01047 | 0.57 | 0.28 |
| P10 | 1 | 305 | 1 | 0.76982 | 0 | 1 | 1 | 257 | 1 | 0.76805 | 0.00177 | 0.93 | 0.93 |
| P11 | 8 | 331 | 8 | 0.80999 | 0 | 8 | 1 | 334 | 5.97 | 0.77999 | 0.03 | 1.07 | 0.13 |
| P12 | 5 | 401 | 4.57 | 0.78311 | 0.00080 | 4.57 | 0.91 | 286 | 3.87 | 0.78117 | 0.00273 | 3.8 | 0.76 |
| P13 | 6 | 356 | 6 | 0.82175 | 0.00001 | 5.9 | 0.98 | 334 | 5.93 | 0.82170 | 0.00006 | 5.63 | 0.94 |
| P14 | 7 | 306 | 6.97 | 0.75140 | 0.00025 | 6.73 | 0.96 | 167 | 6.27 | 0.75135 | 0.0003 | 5.77 | 0.82 |
| P15 | 6 | 305 | 5.17 | 0.81071 | 0.00159 | 5.13 | 0.86 | 161 | 5.07 | 0.80956 | 0.00274 | 4.4 | 0.73 |
| P16 | 15 | 316 | 11.13 | 0.86655 | 0.00203 | 8.9 | 0.59 | 145 | 8.03 | 0.85877 | 0.00981 | 3.27 | 0.22 |
| P17 | 9 | 403 | 8.97 | 0.89432 | 0 | 8.97 | 1 | 327 | 8.80 | 0.89431 | 0.00001 | 8.57 | 0.95 |
| P18 | 3 | 326 | 2.83 | 0.84459 | 0.00079 | 2.83 | 0.94 | 295 | 2.20 | 0.84149 | 0.00389 | 2.17 | 0.72 |
| P19 | 20 | 387 | 18.33 | 0.79192 | 0.00005 | 15.20 | 0.76 | 377 | 13.87 | 0.78595 | 0.00602 | 2.87 | 0.14 |
| P20 | 13 | 469 | 12.73 | 0.90341 | 0.00166 | 10 | 0.77 | 408 | 10.10 | 0.89067 | 0.01441 | 2.5 | 0.19 |
| P21 | 7 | 327 | 5.9 | 0.84067 | 0.00044 | 5.7 | 0.81 | 238 | 4.57 | 0.83174 | 0.00937 | 0.83 | 0.12 |
| P22 | 4 | 408 | 4 | 0.82795 | 0 | 4 | 1 | 369 | 4 | 0.82795 | 0 | 4 | 1 |
| P23 | 8 | 320 | 7.93 | 0.75296 | 0 | 7.93 | 0.99 | 343 | 6.8 | 0.75257 | 0.00039 | 6.47 | 0.81 |
| P24 | 6 | 431 | 6 | 0.81481 | 0 | 6 | 1 | 153 | 6.83 | 0.81034 | 0.00447 | 2.7 | 0.45 |
| P25 | 7 | 386 | 6.87 | 0.90198 | 0 | 6.63 | 0.95 | 350 | 5.63 | 0.90197 | 0 | 3.4 | 0.49 |
| P26 | 1 | 357 | 1 | 0.89401 | 0 | 1 | 1 | 220 | 1 | 0.89401 | 0 | 1 | 1 |
| P27 | 4 | 643 | 4 | 0.74469 | 0 | 4 | 1 | 563 | 4 | 0.74066 | 0.00404 | 3.6 | 0.90 |
| P28 | 5 | 700 | 4.93 | 0.88641 | 0 | 4.93 | 0.99 | 577 | 2.10 | 0.88319 | 0.00322 | 2.1 | 0.42 |
| P29 | 1 | 388 | 1 | 0.85228 | 0 | 1 | 1 | 294 | 1 | 0.85228 | 0 | 1 | 1 |
| P30 | 1 | 484 | 1 | 0.84137 | 0 | 1 | 1 | 374 | 1 | 0.84137 | 0 | 1 | 1 |

Tabela 4.2 Resultados (em média) dos algoritmos para 4000 iterações

A Tabela 4.3 mostra a frequência (ou número de *runs*) com que os algoritmos conseguiram obter o conjunto de soluções ótimas na totalidade. *NrO* é o número de soluções ótimas do processo, já referido na Tabela 4.1. Naturalmente, se o número alcançado for igual a 30, o valor da percentagem de soluções ótimas, *%Ótimas* (Tabela 4.2), para esse algoritmo será igual a 1.

| ID | NrO | Vezes | |
|---|---|---|---|
| | | SA | LS |
| P1 | 3 | 20 | 0 |
| P2 | 5 | 30 | 30 |
| P3 | 6 | 30 | 1 |
| P4 | 3 | 24 | 0 |
| P5 | 3 | 30 | 30 |
| P6 | 1 | 30 | 20 |
| P7 | 1 | 30 | 30 |
| P8 | 2 | 30 | 30 |
| P9 | 2 | 16 | 5 |
| P10 | 1 | 30 | 28 |
| P11 | 8 | 30 | 0 |
| P12 | 5 | 17 | 1 |
| P13 | 6 | 27 | 20 |
| P14 | 7 | 28 | 11 |
| P15 | 6 | 4 | 0 |
| P16 | 15 | 0 | 0 |
| P17 | 9 | 29 | 22 |
| P18 | 3 | 25 | 5 |
| P19 | 20 | 0 | 0 |
| P20 | 13 | 11 | 0 |
| P21 | 7 | 13 | 0 |
| P22 | 4 | 30 | 30 |
| P23 | 8 | 28 | 7 |
| P24 | 6 | 30 | 0 |
| P25 | 7 | 22 | 0 |
| P26 | 1 | 30 | 30 |
| P27 | 4 | 30 | 27 |
| P28 | 5 | 28 | 0 |
| P29 | 1 | 30 | 30 |
| P30 | 1 | 30 | 30 |
| **Média** | | **28.5** | **6** |

Tabela 4.3 Frequência de conjuntos ótimos totais obtidos

Com a informação das duas tabelas, é possível analisar o desempenho dos algoritmos a partir de vários pontos de vista. Nenhum indicador de performance sozinho consegue, tipicamente, levar a conclusões completamente fiáveis dos resultados. No problema de otimização em curso, bem como em muitos outros, a razão está no facto de os processos terem características muito diferentes entre eles, tais como: a distância entre os pontos da fronteira de Pareto do conjunto ótimo, o número de pontos na fronteira desse conjunto ótimo, e o tamanho do espaço de soluções (complexidade) do processo. Esses fatores levam à escolha das seguintes métricas na análise dos resultados:

- Uso da métrica *Hypervolume* (colunas *Área* e *Dif*). É determinado o valor da área (volume, se o problema contivesse mais do que dois objetivos) do polígono definido pelos pontos contidos na fronteira de Pareto, assumindo que a origem é o ponto (0,0) do plano dos objetivos Fiabilidade/Contraparte do custo e os eixos nesse quadrante. Quanto mais próximo o valor da área de um conjunto aproximado estiver do valor da área do conjunto ótimo, melhor a prestação do respetivo algoritmo meta-heurístico.
- Percentagem de soluções ótimas (coluna *% Ótimas*) do conjunto aproximado pelo algoritmo contidas no conjunto ótimo. Se os conjuntos contiverem exatamente os mesmos elementos, o valor *% Ótimas* será igual a um.
- Comparação entre os valores do tamanho do conjunto ótimo (coluna *NrO*), e o número de soluções no conjunto aproximado, (coluna *NrOE*). A comparação recai no facto de os conjuntos ótimos dos processos testados terem tamanhos muito variados. Enquanto que seis dos 30 processos têm apenas uma solução ótima, existem três – P16, P19 e P20 – que têm 15, 20 e 13 soluções ótimas.
- Frequência/número de *runs* com que o algoritmo alcançou o conjunto ótimo na totalidade. Ao contrário das métricas anteriormente mencionadas, este não é um valor médio, mas sim a contabilização do número de vezes que o algoritmo encontrou todas as soluções desejadas.

**Tempo dos *runs***

Começamos por observar o tempo médio de execução dos dois algoritmos e reparamos que na maioria dos processos, o SA demorou sempre mais que o LS, como seria expectável. Uma das razões é pela quantidade de computação a que são sujeitos em cada iteração. Uma iteração do LS, na sua essência, consiste em gerar uma nova solução e, eventualmente, percorrer o atual conjunto aproximado. Uma iteração do SA, além de fazer o mesmo que o LS, efetua a estratégia de retorno à base (ou gera uma mutação total) ou calcula uma probabilidade de aceitação (baseada nos dados atuais da execução) usada na decisão de ocorrer, ou não, uma transição de estado. A segunda razão é o número de iterações de cada um dos algoritmos. O SA realizará sempre o número de iterações imposto no início da execução, mas o LS poderá terminar antes de alcançar esse valor se atingir primeiro o número estipulado de iterações estáveis consecutivas.

**Eficácia do LS**

O LS mostrou uma eficácia muito abaixo do SA. Em apenas 12 dos 30 processos, o LS obteve uma percentagem de soluções ótimas superior a 80%, enquanto que em vários dos restantes processos o valor foi bastante inferior a esse. Podemos admitir a hipótese de o limite de iterações estáveis consecutivas ter sido atingido, impossibilitando o LS de, eventualmente, descobrir soluções pertencentes ao conjunto ótimo. No entanto, pelos resultados observados, tudo aponta para que fosse necessário aumentar radicalmente os dois parâmetros para que este pudesse, ainda que sem garantias, encontrar mais soluções ótimas ou, pelo menos, adicionar novas soluções aproximadas ao seu conjunto. Se as novas soluções do conjunto não fossem ótimas, apenas o valor da coluna Área aumentaria, o que seria uma melhoria pouco relevante, comparada ao aumento de *% Ótimas*. É, por isso, um palpite pouco seguro assumir que o LS conseguiria melhorar os resultados de forma substancial, além de que o tempo de computação se iria tornar muito mais dispendioso. Em todo o caso, este algoritmo foi suficiente para alcançar resultados apreciáveis em alguns dos processos com complexidade superior a 30 milhões, tais como P8, P10, P13, P14 e P26.

**Processos com apenas uma solução ótima**

Ambos os algoritmos apresentaram eficácia nos processos com apenas uma solução ótima (6, 7, 10, 26, 29, 30). O SA obteve em todas os *runs* a solução única de todos esses processos. O LS só a encontrou menos vezes no P6 – 20 vezes, não deixando de ser um bom valor – sendo que nos restantes cinco conseguiu sempre um valor igual ou muito próximo de 30. Refira-se que, ao ordenarmos os processos por ordem crescente de complexidades, vemos que três dos seis processos mencionados encontram-se acima do meio da tabela. O P26 é o processo com a sétima complexidade mais elevada, e P10 com a terceira, tendo um espaço de procura de mais de 50 milhões de soluções. Embora a execução de ambos os algoritmos se iniciasse sempre com uma solução resultante da alteração de todos os seus elementos mutáveis, relativamente aos do modelo inicial do processo (conceito de mutação total), os algoritmos revelaram-se capazes de chegar à solução ótima.

**Frequência de soluções ótimas encontradas na totalidade**

Pela Tabela 4.3 vemos que, no caso do SA, este conseguiu alcançar por 20 vezes, ou mais (pelo menos 67% das vezes), o conjunto ótimo em 23 processos. Pelo contrário, para o P15, só o conseguiu quatro vezes, e para o P16 e P19 nenhuma vez. O LS nunca obteve o conjunto ótimo para nenhum dos três processos (P15, P16 e P19). No entanto, estes valores não invalidam, só por si, um desempenho razoável das meta-heurísticas, nomeadamente do SA:

- No P15, o SA obteve uma média de 5.13 soluções ótimas das seis no total, resultando numa percentagem média de 0.86.
- No P19, o SA conseguiu o valor médio 15.2 de 20 soluções ótimas, o que perfaz 0.76.
- O processo P16 foi (não só de entre os três referidos, mas sim dos 30 processos) aquele em que o SA obteve a percentagem mais pequena, 0.59. Embora seja um valor muito inferior à média geral das percentagens por ele obtidas, 0.936, o SA conseguiu uma média de 8.9 soluções ótimas das 15 possíveis.

O que as observações anteriores mostram é que embora um algoritmo possa nunca conseguir atingir o conjunto ótimo na totalidade, tal não invalida a sua boa prestação. O facto de P16 e P19 serem, de entre todos os processos, aqueles com maior *NrO*, faz com

que analisar somente os valores da coluna *% Ótimas* não seja suficiente para concluir a verdadeira qualidade do algoritmo. A decisão correta, além de se observar a coluna referida, é ver o quão distante o valor da coluna *NrOE* se encontra do valor máximo *NrO*.

**Uso insuficiente da métrica *Hypervolume***

A área do conjunto aproximado não nos permite total fiabilidade na interpretação dos resultados. Um exemplo notório deu-se na execução do LS sob o processo P25. Embora a diferença da área do conjunto aproximado para o conjunto ótimo indique o valor zero, o algoritmo não alcançou a totalidade do conjunto ótimo em nenhum dos *runs*. É, aliás, visível que, na média das percentagens de soluções ótimas encontradas, este ficou-se pelo valor 0.49. A razão do valor da área nos induzir a erro tem a ver com os próprios valores do conjunto ótimo do processo, nomeadamente do objetivo da fiabilidade, como se vê na Figura 4.5. À exceção da última solução, a diferença entre os valores de fiabilidade só se fazem notar na sexta casa decimal. Este processo é, aliás, aquele em que os valores de fiabilidade se encontram mais próximos uns dos outros. A grande maioria dos restantes processos apresentam diferenças desses valores entre a primeira e a terceira casa decimal. Ainda assim, em 19 dos 30 *runs*, o algoritmo encontrou quatro a cinco soluções ótimas, de sete possíveis.

```
Reliability, Cost Inverse
0.993002288759,0.8708333333333333
0.993002209062,0.8770833333333333
0.993001923953,0.8791666666666667
0.993001577679,0.89375
0.993001497982,0.9
0.993001212874,0.9020833333333333
0.992977842294,0.9083333333333333
```

Figura 4.5 Conjunto ótimo do processo P25

São exemplos como o anterior que mostram como, nem sempre, se deve realizar a análise apenas com base numa métrica, neste caso a do *Hypervolume*. É aconselhável, sim, conciliar os resultados de vários indicadores, porque resultará sempre numa análise mais fidedigna da eficácia dos algoritmos.

**Influência dos valores de *NrO* e de complexidade**

Uma das assunções possíveis antes da execução dos algoritmos seria dizer que, à medida que a complexidade de um processo aumenta, pior seriam os resultados encontrados. De facto, entre alguns dos piores resultados do SA, estão P19 e P20, os processos com o maior espaço de soluções. Além disso, também nos processos com as complexidades mais pequenas, P27, P28 e P29, o algoritmo obteve excelentes resultados. No entanto, em P8, P10 e P11 (os três processos de maior complexidade a seguir a P19 e P20), o SA obteve bons resultados, muito acima dos conseguidos em P9 e P15, que têm complexidades bem menores. Não é, por isso, completamente fiável afirmar que a complexidade do processo, só por si, influencie a eficácia do SA.

Comparar processos com valores idênticos de complexidade e *NrO* também não revela a existência de um padrão na procura. P4 e P9 têm um igual tamanho de espaço de soluções, e têm três e duas soluções ótimas, respetivamente. E, no entanto, o SA revelou uma percentagem de 93% de soluções ótimas em P4 e apenas de 75% em P9. Além disso, também em P4, o algoritmo encontrou o conjunto ótimo total oito vezes a mais que em P9, como se observa pelos valores de *Freq*.

A observação mais aceitável seria admitir a possibilidade de que, quanto maior a complexidade e o número de soluções ótimas, pior tenderia a ser a qualidade da procura. Efetivamente, 16, 19 e 20, que são os três processos com maior *NrO*, e cujas complexidades estão entre as sete maiores, são dos processos com pior qualidade por parte do SA. No entanto, P9, além dos processos anteriores, foi onde o algoritmo teve um dos piores resultados, embora tenha uma complexidade muito mais pequenas que a dos anteriores, e apenas duas soluções ótimas. Os casos referidos mostram a dificuldade em estabelecer um padrão de qualidade dos algoritmos, de acordo com as características dos processos testados.

**Número de comunicações das soluções encontradas**

Podemos ainda analisar as soluções quanto ao número de comunicações ocorridas entre diferentes participantes, como se vê pela Tabela 4.4. *NrC* (da tabela 4.1) é o número de comunicações do modelo original do processo. *NrC Médio* calcula para os 30 *runs* de cada algoritmo o número médio de comunicações realizadas em cada conjunto de soluções aproximadas. Foram poucos os processos (apenas cinco) onde houve uma variação na média de comunicações entre os algoritmos. Um dos casos notórios foi no

processo 21, onde as soluções encontradas pelo LS têm, em média, 3 comunicações a mais que as do SA. Os resultados da tabela 4.2 ajudam a entender essa diferença: o SA conseguiu uma média de 81% das soluções ótimas, e o LS apenas 12%.

No entanto, o que os números da tabela melhor transmitem foi a clara redução de comunicações, em ambos os algoritmos, resultante das soluções descentralizadas encontradas.

| ID | NrC | NrC Médio | |
|---|---|---|---|
| | | SA | LS |
| P1 | 24 | 6 | 6 |
| P2 | 17 | 6 | 6 |
| P3 | 22 | 10 | 10 |
| P4 | 20 | 11 | 11 |
| P5 | 22 | 11 | 11 |
| P6 | 19 | 6 | 6 |
| P7 | 15 | 9 | 9 |
| P8 | 23 | 14 | 14 |
| P9 | 16 | 12 | 12 |
| P10 | 19 | 11 | 11 |
| P11 | 23 | 11 | 12 |
| P12 | 20 | 10 | 10 |
| P13 | 19 | 9 | 9 |
| P14 | 19 | 9 | 8 |
| P15 | 19 | 9 | 9 |
| P16 | 20 | 11 | 11 |
| P17 | 22 | 11 | 11 |
| P18 | 16 | 9 | 9 |
| P19 | 23 | 13 | 13 |
| P20 | 21 | 13 | 13 |
| P21 | 23 | 8 | 11 |
| P22 | 19 | 9 | 9 |
| P23 | 19 | 14 | 13 |
| P24 | 20 | 9 | 9 |
| P25 | 20 | 12 | 11 |
| P26 | 24 | 10 | 10 |
| P27 | 27 | 16 | 16 |
| P28 | 20 | 10 | 10 |
| P29 | 16 | 6 | 6 |
| P30 | 23 | 11 | 11 |

Tabela 4.4 Número médio de comunicações nas soluções encontradas

**Realização de novos testes no SA**

Em seguida, foram executados novos testes, com os parâmetros a serem alterados para um número de 20000 iterações. O LS foi excluído, pois revelou uma performance muito aquém da conseguida pelo SA, em muitos dos processos. Desta vez, foram apenas considerados os processos onde o SA teve os piores resultados, nomeadamente aqueles em que *% Ótimas* foi menor que 90% e/ou a frequência com que chegou ao conjunto total de soluções ótimas ficou abaixo dos 20 *runs*. Os restantes 22 processos não foram testados novamente porque assumiu-se que, se o algoritmo obteve uma eficácia apreciável com 4000 iterações, seria desnecessário aumentar as iterações para esses casos. A Tabela 4.5 mostra os resultados da execução, igualmente com 30 *runs* aplicados a cada processo. Os seis primeiros campos são iguais aos da Tabela 4.2. A coluna *Freq* significa o mesmo que na Tabela 4.3.

| ID | T(ms) | NrS | Área | Dif | NrOE | % Ótimas | Freq |
|---|---|---|---|---|---|---|---|
| P1 M | 1422 | 3.00 | 0.88972 | 0.00000 | 3.00 | 1.00 | 30 |
| P9 M | 1578 | 2.03 | 0.74483 | 0.00004 | 1.97 | 0.98 | 29 |
| P12 M | 2366 | 4.83 | 0.78360 | 0.00031 | 4.83 | 0.97 | 25 |
| P15 M | 1643 | 5.40 | 0.81116 | 0.00114 | 5.40 | 0.90 | 12 |
| P16 M | 1542 | 13.30 | 0.86829 | 0.00029 | 13.13 | 0.88 | 14 |
| P19 M | 1908 | 19.33 | 0.79196 | 0.00001 | 19.33 | 0.97 | 11 |
| P20 M | 2160 | 13.00 | 0.90507 | 0.00001 | 11.50 | 0.88 | 22 |
| P21 M | 1607 | 7.00 | 0.84111 | 0.00000 | 7.00 | 1.00 | 30 |

Tabela 4.5 Resultados (em média) do SA para 20000 iterações

Os resultados mostram melhorias significativas em todos os processos. Em P1 e P21, o aumento de iterações foi suficiente para que o SA obtivesse o ótimo, obtendo os valores máximos em *% Ótimas* e *Freq*. Em P9 e P12, os resultados também ficaram muito próximos dos valores máximos. Embora em P19, a procura tenha obtido o conjunto ótimo apenas por 11 vezes, há-que relembrar que este é o processo com a maior complexidade e maior número de soluções ótimas, de entre todas as instâncias geradas. E, ainda assim, *NrOE* ficou muito próximo do valor máximo, 20, assim como *% Ótimas* do valor um.

Por fim, notar que P15 foi onde o algoritmo teve a melhoria menos significativa, ainda que tendo valores de complexidade e *NrO* abaixo de P16 e P20. Processos esses, onde o aumento dos valores das métricas foi claramente acima do aumento em P15.

# Capítulo 5 Conclusão

Neste capítulo são apresentadas as principais contribuições dadas por este trabalho, bem como as conclusões que se retiram dos resultados observados. São ainda apresentadas sugestões de melhorias que podem ser feitas numa futura iteração.

## 5.1 Trabalho desenvolvido e conclusões

Este trabalho contribuiu com a continuação de uma ideia, já iniciada, que tem como objetivo desenvolver métodos automáticos de descentralização e otimização de processos de negócio dependentes da Internet das Coisas. Muitos destes processos apresentam, ainda hoje, uma arquitetura centralizada onde ocorre um grande número de comunicações entre o sistema central e os aparelhos *IoT*, e que pode resultar em custos elevados para as organizações. Um método previamente disponível permite gerar instâncias pseudo-aleatórias de processos, modelados na linguagem BPMN.

Neste trabalho procurou-se avançar a otimização da descentralização com as seguintes contribuições:

- Foram realizados testes e correções ao *software* já existente, e efetuadas melhorias ao nível da interface.
- Foi estabelecido um problema de otimização que procura obter as soluções descentralizadas de processos que melhor correspondem aos dois objetivos em simultâneo: maximizar o valor de fiabilidade geral e minimizar o custo de comunicação entre participantes.
- Foram desenvolvidos algoritmos que visam responder ao problema de otimização referido. Um deles tem natureza determinística e realiza uma comparação, de forma exaustiva, de todas as soluções descentralizadas de um processo. Os outros dois algoritmos são métodos meta-heurísticos e pretende-se que sejam uma alternativa válida ao algoritmo exaustivo. A qualidade das meta-heurísticas foi avaliada através de várias métricas, tendo por base de comparação os resultados obtidos pela procura exaustiva.

Os resultados do trabalho mostram dois tipos diferentes de métodos válidos na abordagem ao problema de otimização estabelecido. Sempre que o tempo requerido de

computação for comportável, o algoritmo de busca exaustiva, desenvolvido neste trabalho, pode ser usado na procura das soluções que melhor correspondem às variáveis de decisão aqui consideradas, fiabilidade e custo de comunicação, aquando da descentralização do processo. A vantagem desse método é que, seguramente, encontrará as soluções ótimas do problema. Em processos de maiores dimensões, onde é natural ocorrer um aumento de participantes, bem como de atividades transferíveis para a *pool IoT*, o uso da busca exaustiva pode tornar-se dispendioso. Para esse caso, os algoritmos meta-heurísticos implementados mostraram ser uma alternativa válida na procura de soluções adequadas. Embora a descoberta das soluções ótimas não seja garantida através desses métodos, estes revelaram uma boa qualidade de resultados. O *Simulated Annealing*, principalmente, conseguiu obter o conjunto ótimo em várias das instâncias utilizadas. Nos restantes casos, tipicamente nos processos com maior valor de complexidade e número de soluções ótimas, o algoritmo conseguiu, ainda assim, uma boa aproximação a esse conjunto. O *Local Search* também obteve os conjuntos ótimos nalgumas das instâncias, embora com uma eficácia bem inferior à do *Simulated Annealing*. O facto de ambos os algoritmos também beneficiarem de um tempo de computação muito reduzido possibilita que o número de iterações seja aumentado em futuros processos com maior complexidade do que aqueles possíveis de serem testados neste trabalho.

## 5.2 Trabalho futuro

Entre os aspetos a melhorar num trabalho futuro encontram-se:

- Considerar processos de negócio de maior dimensão, nomeadamente ao nível de elementos que possam ser executados por mais do que um participante. O aumento desse tipo de elementos resultará num maior número de alternativas descentralizadas por analisar.
- Integrar mecanismos de minimização do risco, na descentralização dos processos. Os elementos contidos num processo de negócio podem conter pontos de falha que, dependendo do nível de impacto, podem comprometer a execução geral do processo, e consequentemente prejudicar a organização. Determinar o risco de cada um dos elementos, de acordo com os respetivos valores de fiabilidade e impacto, será um dos caminhos possíveis para o cálculo do risco total do processo.

- Pelas razões acima referidas, os algoritmos utilizados poderão ter de sofrer alterações, tanto ao nível do número total de iterações, como no tipo de estratégia usada na procura. Novos algoritmos poderão ter de ser implementados devido a fatores decisivos, como o aumento do espaço de soluções ou a introdução de novos objetivos no problema como, por exemplo, o risco geral do processo. Algoritmos evolutivos, como o NSGA-II [8] e SPEA-2 [32], são dos mais utilizados para problemas com dois ou mais objetivos, e a sua possível escolha deve ser tida em conta.
- Introdução de novos indicadores de qualidade dos algoritmos meta-heurísticos na sua análise.

# Bibliografia


[1] Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. Computer networks, 54(15), 2787-2805.

[2] Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. Natural Computing, 8(2), 239-287.

[3] Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM computing surveys (CSUR), 35(3), 268-308.

[4] BPMN2 Modeler. (n.d.). Eclipse Foundation. https://www.eclipse.org/bpmn2-modeler/ [acedido: 19-Nov-2020]

[5] Coello, C. C. (1999, July). An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406) (Vol. 1, pp. 3-13). IEEE.

[6] Cardoso, A. J. S. (2002). Quality of service and semantic composition of workflows (Doctoral dissertation, University of Georgia).

[7] Domingos, D., Martins, F., Martinho, R., & Silva, M. (2010). Ad-hoc changes in IoT-aware business processes. In 2010 Internet of Things (IOT) (pp. 1-7). IEEE.

[8] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), 182-197.

[9] Domingos, D., Respício, A. & Martinho, R. (2016). Using Resource Reliability in BPMN Processes. Procedia Computer Science. 100.

[10] Domingos, D., Respício, A., Martins, F., & Melo, B. (2019). Automatic Decomposition of IoT Aware Business Processes – a Pattern Approach. Procedia Computer Science. 164. 313-320. 10.

[11] Fernández-Vargas, J. A., Bonilla-Petriciolet, A., Rangaiah, G. P., & Fateen, S. E. K. (2016). Performance analysis of stopping criteria of population-based metaheuristics for global optimization in phase equilibrium calculations and modeling. Fluid Phase Equilibria, 427, 104-125.

[12] Goldbarg, M. C., & Luna, H. P. L. (2005). Otimização combinatória e programação linear: modelos e algoritmos. Elsevier.

[13] Internet of Things. (n.d.). IERC-European Research Cluster on the Internet of Things. http://www.internet-of-things-research.eu/about_iot.htm [acedido: 29-Nov-2020]

[14] Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by Simulated Annealing. Science, 220, 671 - 680.

[15] Knowles, J., & Corne, D. (2002, May). On metrics for comparing nondominated sets. In Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600) (Vol. 1, pp. 711-716). IEEE.

[16] Leloglu, E. (2016). A review of security concerns in Internet of Things. Journal of Computer and Communications, 5(1), 121-136.

[17] Martinho, R., & Domingos, D. (2014). Quality of information and access cost of IoT resources in BPMN processes. Procedia Technology, 16, 737-744.

[18] Martins, F., Domingos, D., & Vitoriano, D. (2019). Automatic Decomposition of IoT Aware Business Processes with Data and Control Flow Distribution. In ICEIS (2) (pp. 516-524).

[19] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6), 1087-1092.

[20] Miorandi, D., Sicari, S., De Pellegrini, F., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. Ad hoc networks, 10(7), 1497-1516.

[21] Object Management Group (OMG). (2011) Business Process Model and Notation (BPMN) Version 2.0.

[22] Osyczka, A. (1985). Multicriteria optimization for engineering design. In Design optimization (pp. 193-227). Academic Press.

[23] Pettinger, T. (n.d.). Pareto efficiency. Economics Help. https://www.economicshelp.org/blog/glossary/pareto-efficiency/ [acedido: 29-Jan-2020]

[24] Respício, A., & Domingos, D. (2015). Reliability of BPMN business processes. Procedia Computer Science, 64, 643-650.

[25] Respício, A., Martinho, R., & Domingos, D. (2017). Reliability of AAL Systems Modeled as BPMN Business Processes. In International Conference on Enterprise Information Systems (pp. 535-550). Springer, Cham.

[26] Serafini, P. (1994). Simulated annealing for multi objective optimization problems. In Multiple criteria decision making (pp. 283-292). Springer, New York, NY.

[27] Suman, B., & Kumar, P. (2006). A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the operational research society, 57(10), 1143-1160.

[28] Suppapitnarm, A., Seffen, K. A., Parks, G. T., & Clarkson, P. J. (2000). A simulated annealing algorithm for multiobjective optimization. Engineering Optimization, 33(1), 59-85.

[29] Yang, Y., Wu, L., Yin, G., Li, L., & Zhao, H. (2017). A survey on security and privacy issues in Internet-of-Things. IEEE Internet of Things Journal, 4(5), 1250-1258.

[30] Zavazava, C. (2015, March). ITU work on Internet of Things. In Presentation at ICTP Workshop.

[31] Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary computation, 8(2), 173-195.

[32] Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. TIK-report, 103.

[33] Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. IEEE transactions on Evolutionary Computation, 3(4), 257-271.

# Anexo A – Instâncias pseudo-aleatórias

Conteúdo da Tabela 4.1 ordenando as linhas por ordem crescente de complexidade das instâncias

| ID | NrP | NrE | NrEM | NrC | Comp | T(min) | NrO | A |
|---|---|---|---|---|---|---|---|---|
| 28 | 2 | 38 | 19 | 20 | 524288 | 2 | 5 | 0.88641 |
| 27 | 2 | 50 | 20 | 27 | 1048576 | 4 | 4 | 0.74469 |
| 29 | 2 | 40 | 20 | 16 | 1048576 | 3 | 1 | 0.85228 |
| 30 | 2 | 40 | 21 | 23 | 2097152 | 7 | 1 | 0.84137 |
| 25 | 3 | 27 | 17 | 20 | 2239488 | 7 | 7 | 0.90198 |
| 2 | 3 | 25 | 16 | 17 | 2519424 | 6 | 5 | 0.81554 |
| 7 | 4 | 22 | 14 | 15 | 4718592 | 11 | 1 | 0.72252 |
| 18 | 3 | 26 | 16 | 16 | 8503056 | 27 | 3 | 0.84538 |
| 3 | 3 | 25 | 19 | 22 | 8957952 | 26 | 6 | 0.69084 |
| 23 | 3 | 27 | 19 | 19 | 8957952 | 26 | 8 | 0.75296 |
| 24 | 3 | 27 | 18 | 20 | 10077696 | 35 | 6 | 0.81481 |
| 4 | 4 | 22 | 16 | 20 | 11943936 | 32 | 3 | 0.73262 |
| 9 | 4 | 23 | 16 | 16 | 11943936 | 33 | 2 | 0.74487 |
| 17 | 3 | 26 | 20 | 22 | 11943936 | 40 | 9 | 0.89432 |
| 5 | 4 | 22 | 15 | 22 | 13436928 | 40 | 3 | 0.82027 |
| 12 | 3 | 25 | 18 | 20 | 15116544 | 50 | 5 | 0.7839 |
| 6 | 4 | 22 | 14 | 19 | 15925248 | 46 | 1 | 0.82201 |
| 15 | 3 | 26 | 19 | 19 | 20155392 | 57 | 6 | 0.8123 |
| 22 | 3 | 27 | 19 | 19 | 20155392 | 77 | 4 | 0.82795 |
| 1 | 3 | 25 | 18 | 24 | 22674816 | 85 | 3 | 0.88972 |
| 13 | 3 | 26 | 19 | 19 | 30233088 | 95 | 6 | 0.82175 |
| 16 | 3 | 26 | 19 | 20 | 30233088 | 88 | 15 | 0.86858 |
| 21 | 3 | 27 | 19 | 23 | 30233088 | 98 | 7 | 0.84111 |
| 26 | 4 | 23 | 17 | 24 | 31850496 | 96 | 1 | 0.89401 |
| 14 | 3 | 26 | 18 | 19 | 34012224 | 98 | 7 | 0.75165 |
| 11 | 3 | 25 | 19 | 23 | 45349632 | 140 | 8 | 0.80999 |
| 8 | 4 | 23 | 16 | 23 | 53747712 | 166 | 2 | 0.78904 |
| 10 | 4 | 23 | 17 | 19 | 56623104 | 160 | 1 | 0.76982 |
| 19 | 3 | 27 | 20 | 23 | 60466176 | 206 | 20 | 0.79197 |
| 20 | 3 | 27 | 20 | 21 | 60466176 | 222 | 13 | 0.90507 |

# Anexo B – Resultados dos algoritmos – instâncias ordenadas por complexidade

| ID | NrO | SA | | | | | | LS | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T(ms) | NrS | Área | Dif | NrOE | % Ótimas | T(ms) | NrS | Área | Dif | NrOE | % Ótimas |
| P28 | 5 | 700 | 4.93 | 0.88641 | 0 | 4.93 | 0.99 | 577 | 2.10 | 0.88319 | 0.00322 | 2.1 | 0.42 |
| P27 | 4 | 643 | 4 | 0.74469 | 0 | 4 | 1 | 563 | 4 | 0.74066 | 0.00404 | 3.6 | 0.90 |
| P29 | 1 | 388 | 1 | 0.85228 | 0 | 1 | 1 | 294 | 1 | 0.85228 | 0 | 1 | 1 |
| P30 | 1 | 484 | 1 | 0.84137 | 0 | 1 | 1 | 374 | 1 | 0.84137 | 0 | 1 | 1 |
| P25 | 7 | 386 | 6.87 | 0.90198 | 0 | 6.63 | 0.95 | 350 | 5.63 | 0.90197 | 0 | 3.4 | 0.49 |
| P2 | 5 | 272 | 5 | 0.81554 | 0 | 5 | 1 | 191 | 5 | 0.81554 | 0 | 5 | 1 |
| P7 | 1 | 268 | 1 | 0.72252 | 0 | 1 | 1 | 295 | 1 | 0.72252 | 0 | 1 | 1 |
| P18 | 3 | 326 | 2.83 | 0.84459 | 0.00079 | 2.83 | 0.94 | 295 | 2.20 | 0.84149 | 0.00389 | 2.17 | 0.72 |
| P3 | 6 | 293 | 6 | 0.69084 | 0 | 6 | 1 | 143 | 5.90 | 0.68482 | 0.00602 | 3.27 | 0.54 |
| P23 | 8 | 320 | 7.93 | 0.75296 | 0 | 7.93 | 0.99 | 343 | 6.8 | 0.75257 | 0.00039 | 6.47 | 0.81 |
| P24 | 6 | 431 | 6 | 0.81481 | 0 | 6 | 1 | 153 | 6.83 | 0.81034 | 0.00447 | 2.7 | 0.45 |
| P4 | 3 | 275 | 2.8 | 0.73105 | 0.00157 | 2.8 | 0.93 | 244 | 1.87 | 0.72062 | 0.012 | 1.33 | 0.44 |
| P9 | 2 | 302 | 2.27 | 0.74375 | 0.00111 | 1.5 | 0.75 | 365 | 2.13 | 0.73440 | 0.01047 | 0.57 | 0.28 |
| P17 | 9 | 403 | 8.97 | 0.89432 | 0 | 8.97 | 1 | 327 | 8.80 | 0.89431 | 0.00001 | 8.57 | 0.95 |
| P5 | 3 | 296 | 3 | 0.82027 | 0 | 3 | 1 | 266 | 3 | 0.82027 | 0 | 3 | 1 |
| P12 | 5 | 401 | 4.57 | 0.78311 | 0.00080 | 4.57 | 0.91 | 286 | 3.87 | 0.78117 | 0.00273 | 3.8 | 0.76 |
| P6 | 1 | 330 | 1 | 0.82201 | 0 | 1 | 1 | 274 | 1.03 | 0.81670 | 0.00531 | 0.67 | 0.67 |
| P15 | 6 | 305 | 5.17 | 0.81071 | 0.00159 | 5.13 | 0.86 | 161 | 5.07 | 0.80956 | 0.00274 | 4.4 | 0.73 |
| P22 | 4 | 408 | 4 | 0.82795 | 0 | 4 | 1 | 369 | 4 | 0.82795 | 0 | 4 | 1 |
| P1 | 3 | 401 | 2.7 | 0.88666 | 0.00306 | 2.67 | 0.89 | 209 | 2 | 0.88035 | 0.00937 | 2 | 0.67 |
| P13 | 6 | 356 | 6 | 0.82175 | 0.00001 | 5.9 | 0.98 | 334 | 5.93 | 0.82170 | 0.00006 | 5.63 | 0.94 |
| P16 | 15 | 316 | 11.13 | 0.86655 | 0.00203 | 8.9 | 0.59 | 145 | 8.03 | 0.85877 | 0.00981 | 3.27 | 0.22 |
| P21 | 7 | 327 | 5.9 | 0.84067 | 0.00044 | 5.7 | 0.81 | 238 | 4.57 | 0.83174 | 0.00937 | 0.83 | 0.12 |
| P26 | 1 | 357 | 1 | 0.89401 | 0 | 1 | 1 | 220 | 1 | 0.89401 | 0 | 1 | 1 |
| P14 | 7 | 306 | 6.97 | 0.75140 | 0.00025 | 6.73 | 0.96 | 167 | 6.27 | 0.75135 | 0.0003 | 5.77 | 0.82 |
| P11 | 8 | 331 | 8 | 0.80999 | 0 | 8 | 1 | 334 | 5.97 | 0.77999 | 0.03 | 1.07 | 0.13 |
| P8 | 2 | 332 | 2 | 0.78904 | 0 | 2 | 1 | 338 | 2 | 0.78904 | 0 | 2 | 1 |
| P10 | 1 | 305 | 1 | 0.76982 | 0 | 1 | 1 | 257 | 1 | 0.76805 | 0.00177 | 0.93 | 0.93 |
| P19 | 20 | 387 | 18.33 | 0.79192 | 0.00005 | 15.2 | 0.76 | 377 | 13.87 | 0.78595 | 0.00602 | 2.87 | 0.14 |
| P20 | 13 | 469 | 12.73 | 0.90341 | 0.00166 | 10 | 0.77 | 408 | 10.10 | 0.89067 | 0.01441 | 2.5 | 0.19 |